Stringovi

Stringovi

- U programskom jeziku Java (kao i u većini drugih) **string** predstavlja sekvencu karaktera.
- Za razliku od nekih drugih jezika gde se stringovi reprezentuju nizovima znakova i slično, za osnovnu implementaciju stringova u Javi zadužena je posebna klasa **`String`** iz paketa **`java.lang.`**
- Klasa **`String`** sakriva internu strukturu podataka kojom se predstavlja sekvenca karaktera čime se:
  - ne dozvoljava direktna manipulacija karakterima stringa, ali zato
  - pruža mnoštvo metoda ne samo za osnovne operacije nad stringovima, već i naprednije procesiranje tekstualnih podataka.

Stringovi

- Klasa **`String`** je **nepromenljiva** (eng. *immutable*) što znači da jednom napravljena instanca klase ne može da se menja, tj. nemoguće je promeniti sadržaj postojećeg stringa u memoriji.

- Ako je rezultat neke operacije drugačiji string, ta instanca klase **`String`** biće iznova napravljena.

- Takođe, klasa **`String`** je **final**, što znači da se ne može naslediti.

Stringovi

- Iako je deo standardnog API-ja klasa **`String`** je jedina „privilegovana“ klasa u Javi, u smislu da uživa specijalan tretman od strane samog programskog jezika. Naime:
- Objekte tipa **`String`** moguće je kreirati bez korišćenja operatora **`new`**:

```
String hello = "Hello World";
```

- Stringovi se mogu spajati korišćenjem operatora + bez poziva specijalnog metoda:

```
String str = "First" + "Second";
```

Metode klase String

- Klasa `String` definiše veliki broj konstruktora za kreiranje novih instanci iz drugih Stringova, nizova znakova, bajtova, itd.
- Klasa `String` uz druge pomoćne klase, pruža mnogo metoda koje omogućavaju lako i efikasno manipulisanje stringovima
- Metode ćemo posmatrati razvrstano u dve kategorije:
  - **Informativne**
  - **Transformativne**

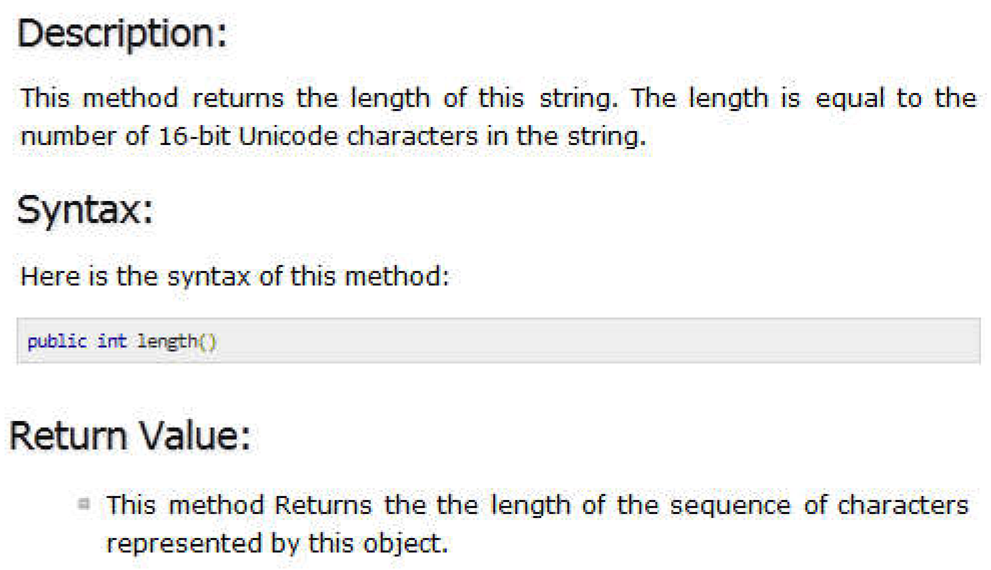
Informativne metode

- **Važnije metode ove vrste su:**
- `int length():` vraća dužinu tekućeg (`this`) stringa
- `int indexOf(...):` nekoliko različitih kombinacija parametara omogućava određivanje pozicije podstringa u tekućem stringu, od početka tekućeg stringa ili od date pozicije nadalje
- `boolean endsWith(String suff):` proverava da li se tekući string završava stringom suff
- `boolean startsWith(String pref):` proverava da li tekući string počinje stringom pref
- `boolean equals(Object anObject):` proverava da li je sadržaj tekućeg stringa jednak sadržaju datog objekta
- `boolean equalsIgnoreCase(String anotherString):` proverava da li je sadržaj tekućeg stringa jednak sadržaju datog stringa, ignorišući razliku između velikih i malih slova
- `Int compareTo(String anotherString):` leksikografsko poređenje stringova

Informativne metode

Informativne metode kao rezultat vraćaju neku informaciju o stringu, kao što je dužina stringa (broj karaktera koje string sadrži), pozicija datog podstringa, itd.

- **Važnije metode ove vrste su:**
- `int length():` vraća dužinu tekućeg (`this`) stringa



Description:

This method returns the length of this string. The length is equal to the number of 16-bit Unicode characters in the string.

Syntax:

Here is the syntax of this method:

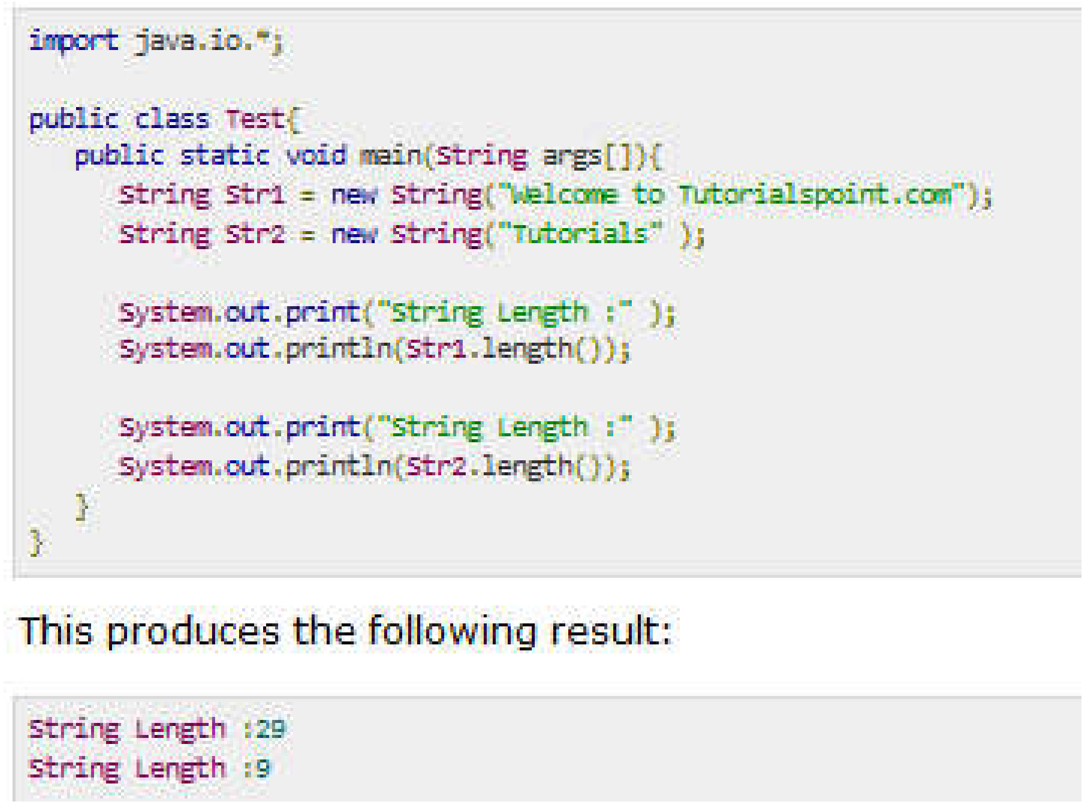
```
public int length()
```

Return Value:

- This method Returns the the length of the sequence of characters represented by this object.

Informativne metode

- `int length():` vraća dužinu tekućeg (`this`) stringa



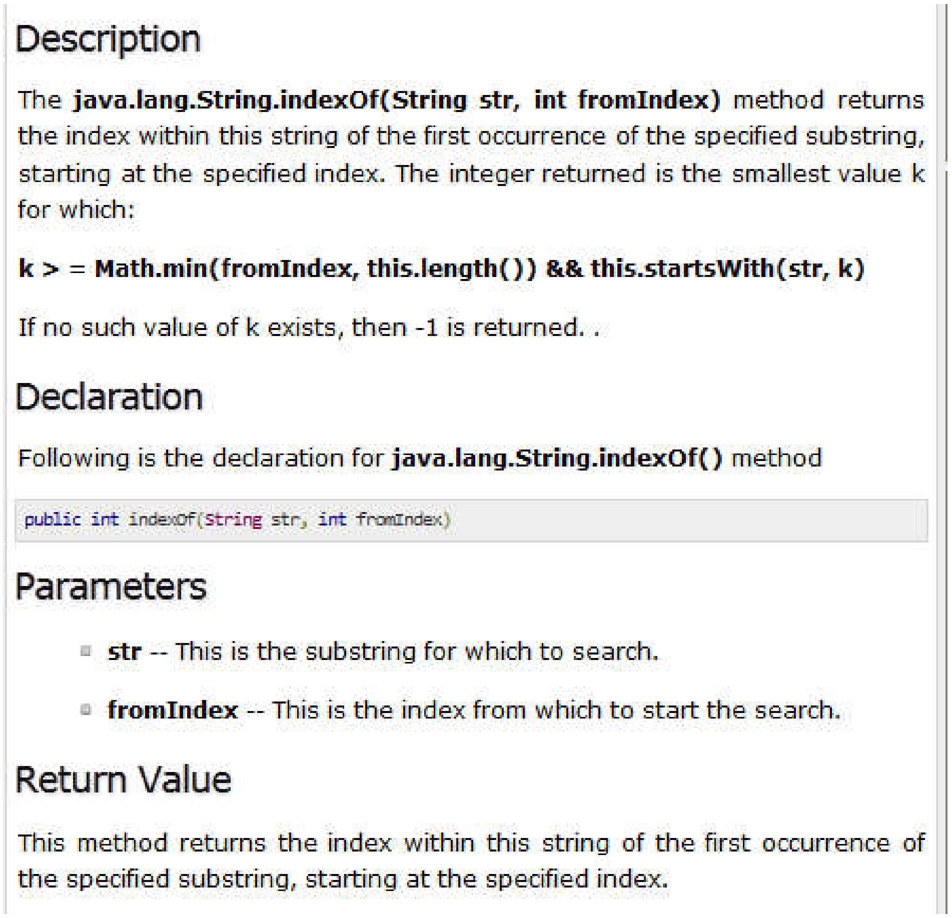
```
import java.io.*;

public class Test{
   public static void main(String args[]){
      String Str1 = new String("Welcome to Tutorialspoint.com");
      String Str2 = new String("Tutorials" );

      System.out.print("String Length :" );
      System.out.println(Str1.length());

      System.out.print("String Length :" );
      System.out.println(Str2.length());
   }
}
```

This produces the following result:

```
String Length :29
String Length :9
```

Informativne metode

- `int indexOf(...)`: nekoliko različitih kombinacija parametara omogućava određivanje pozicije podstringa u tekućem stringu, od početka tekućeg stringa ili od zadate pozicije nadalje



Description

The **java.lang.String.indexOf(String str, int fromIndex)** method returns the index within this string of the first occurrence of the specified substring, starting at the specified index. The integer returned is the smallest value k for which:

k > = Math.min(fromIndex, this.length()) && this.startsWith(str, k)

If no such value of k exists, then -1 is returned. .

Declaration

Following is the declaration for **java.lang.String.indexOf()** method

```
public int indexOf(String str, int fromIndex)
```

Parameters

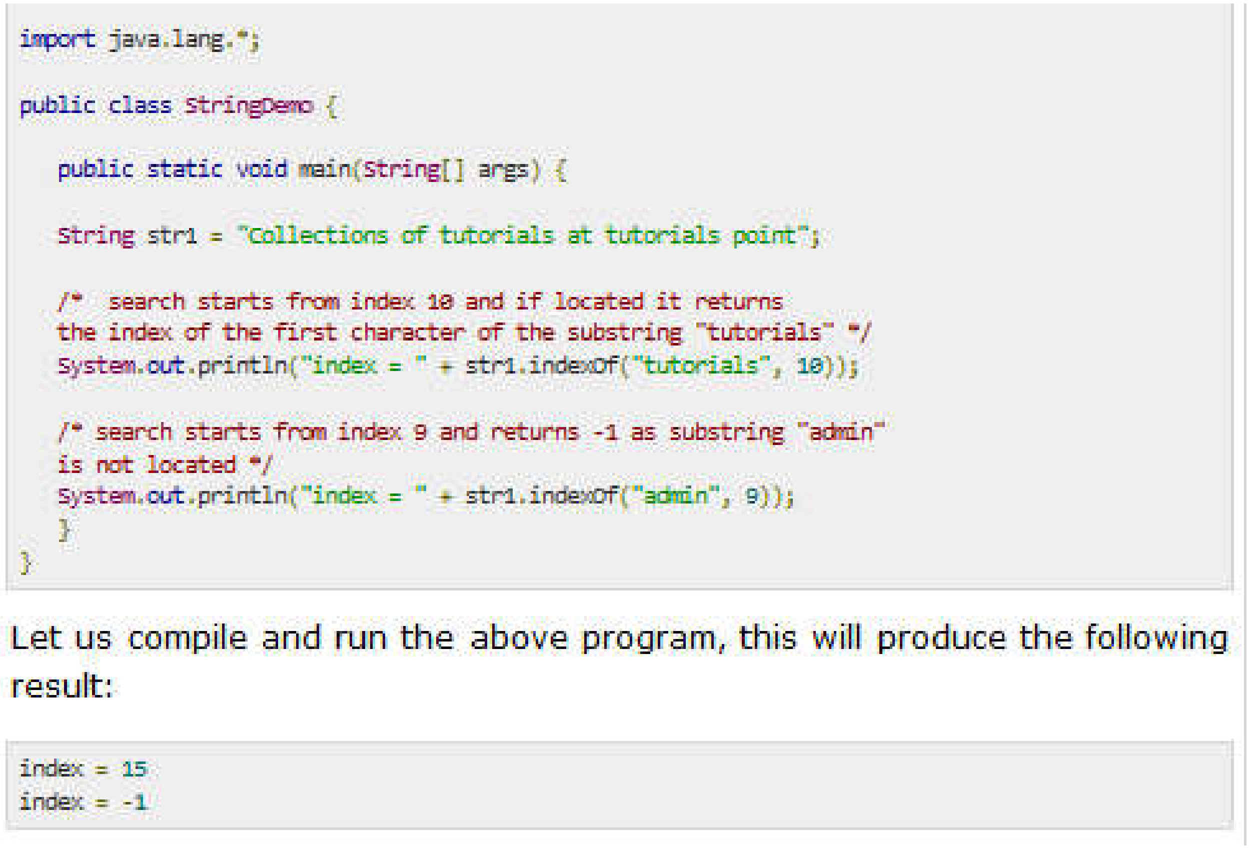
- **str** -- This is the substring for which to search.
- **fromIndex** -- This is the index from which to start the search.

Return Value

This method returns the index within this string of the first occurrence of the specified substring, starting at the specified index.

Informativne metode

- **`int indexOf(...):`** nekoliko različitih kombinacija parametara omogućava određivanje pozicije podstringa u tekućem stringu, od početka tekućeg stringa ili od zadate pozicije nadalje



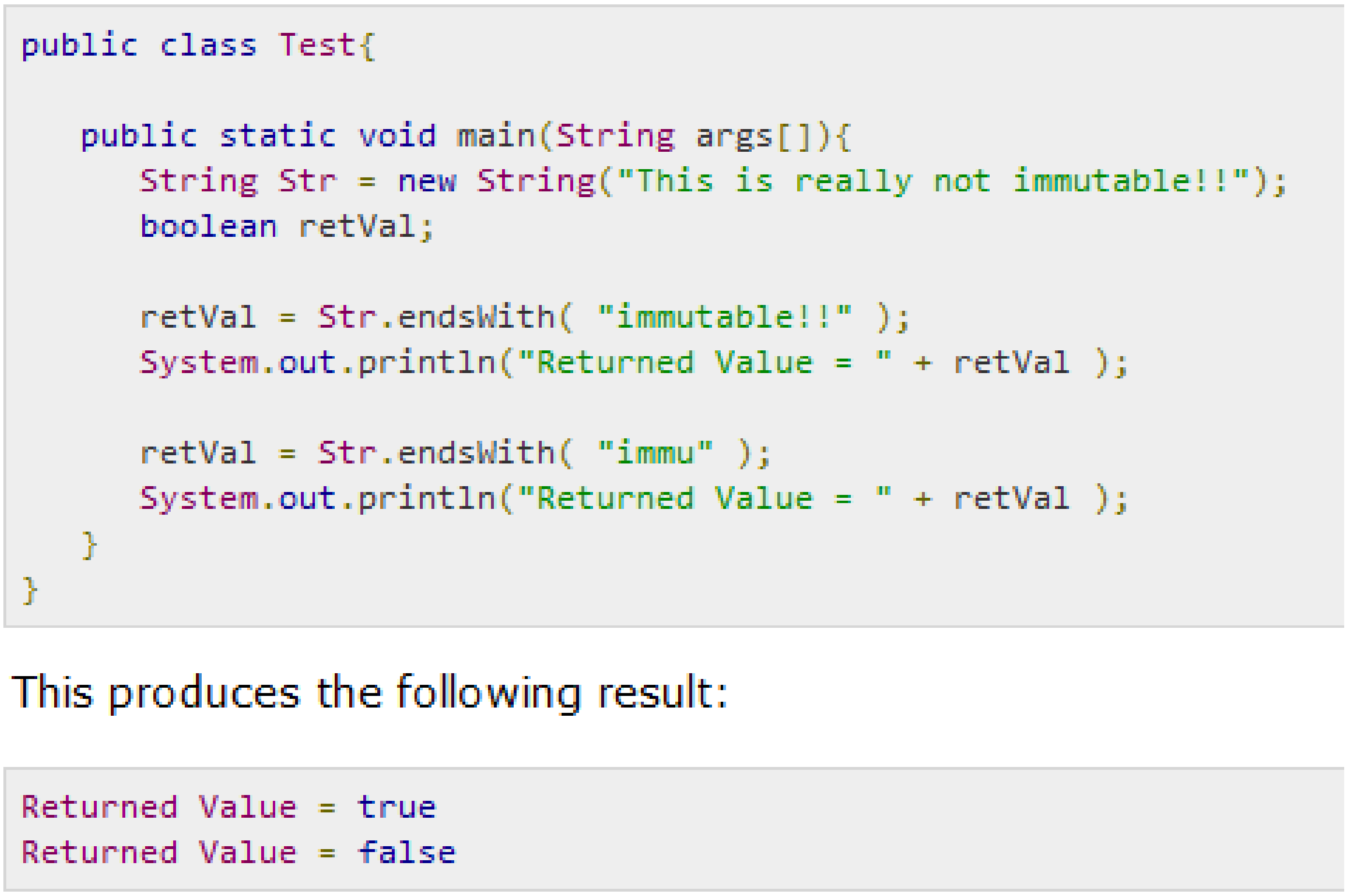
```
import java.lang.*;

public class StringDemo {

   public static void main(String[] args) {

   String str1 = "Collections of tutorials at tutorials point";

   /*  search starts from index 10 and if located it returns
   the index of the first character of the substring "tutorials" */
   System.out.println("index = " + str1.indexOf("tutorials", 10));

   /* search starts from index 9 and returns -1 as substring "admin"
   is not located */
   System.out.println("index = " + str1.indexOf("admin", 9));
   }
}
```

Let us compile and run the above program, this will produce the following result:

```
index = 15
index = -1
```

Informativne metode

- **boolean endsWith(String suff):** proverava da li se tekući string završava stringom suff



```
public class Test{

   public static void main(String args[]){
      String Str = new String("This is really not immutable!!");
      boolean retVal;

      retVal = Str.endsWith( "immutable!!" );
      System.out.println("Returned Value = " + retVal );

      retVal = Str.endsWith( "immu" );
      System.out.println("Returned Value = " + retVal );
   }
}
```

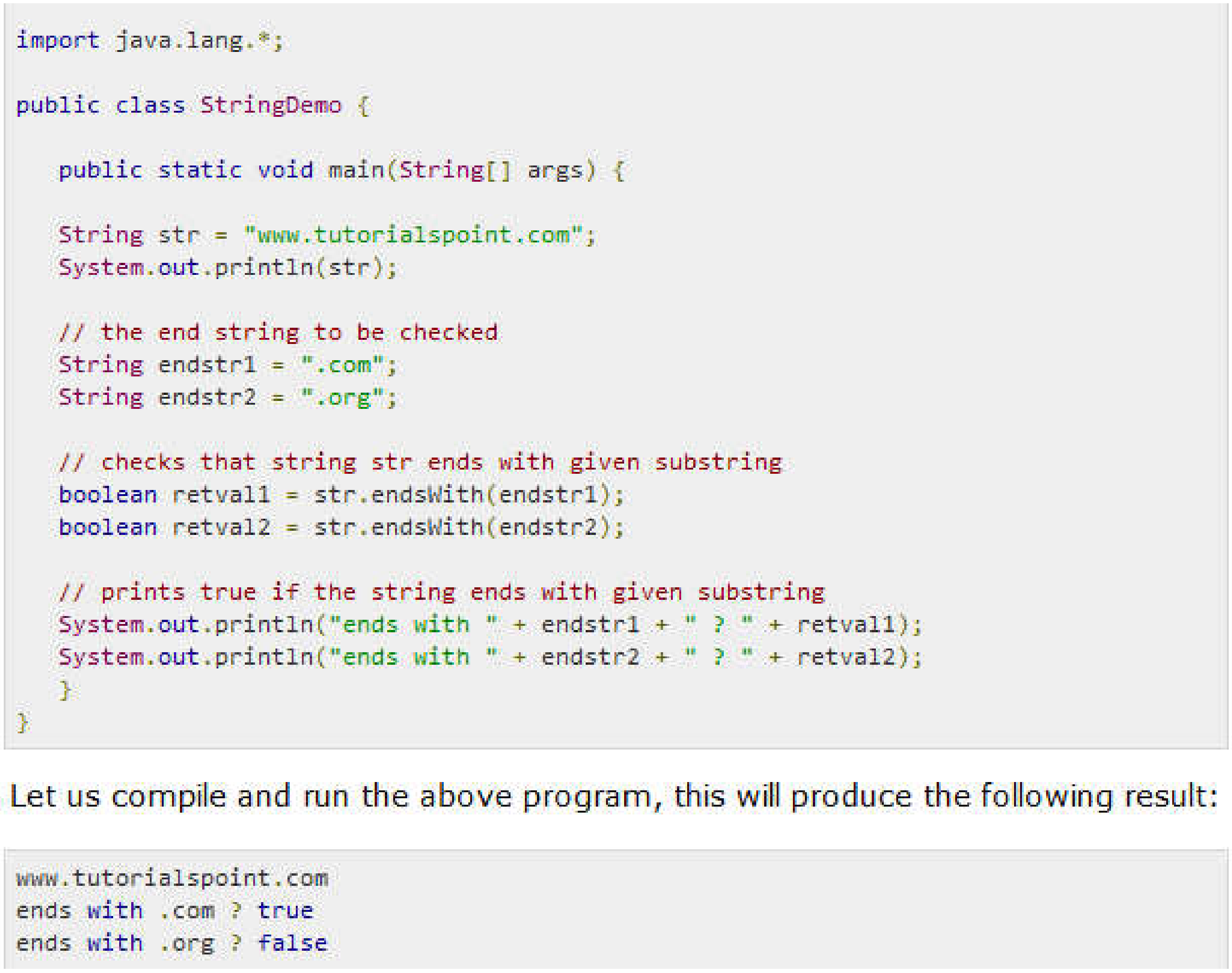
This produces the following result:

```
Returned Value = true
Returned Value = false
```

Stringovi

Informativne metode

- **`boolean endsWith(String suff):`** proverava da li se tekući string završava stringom suff



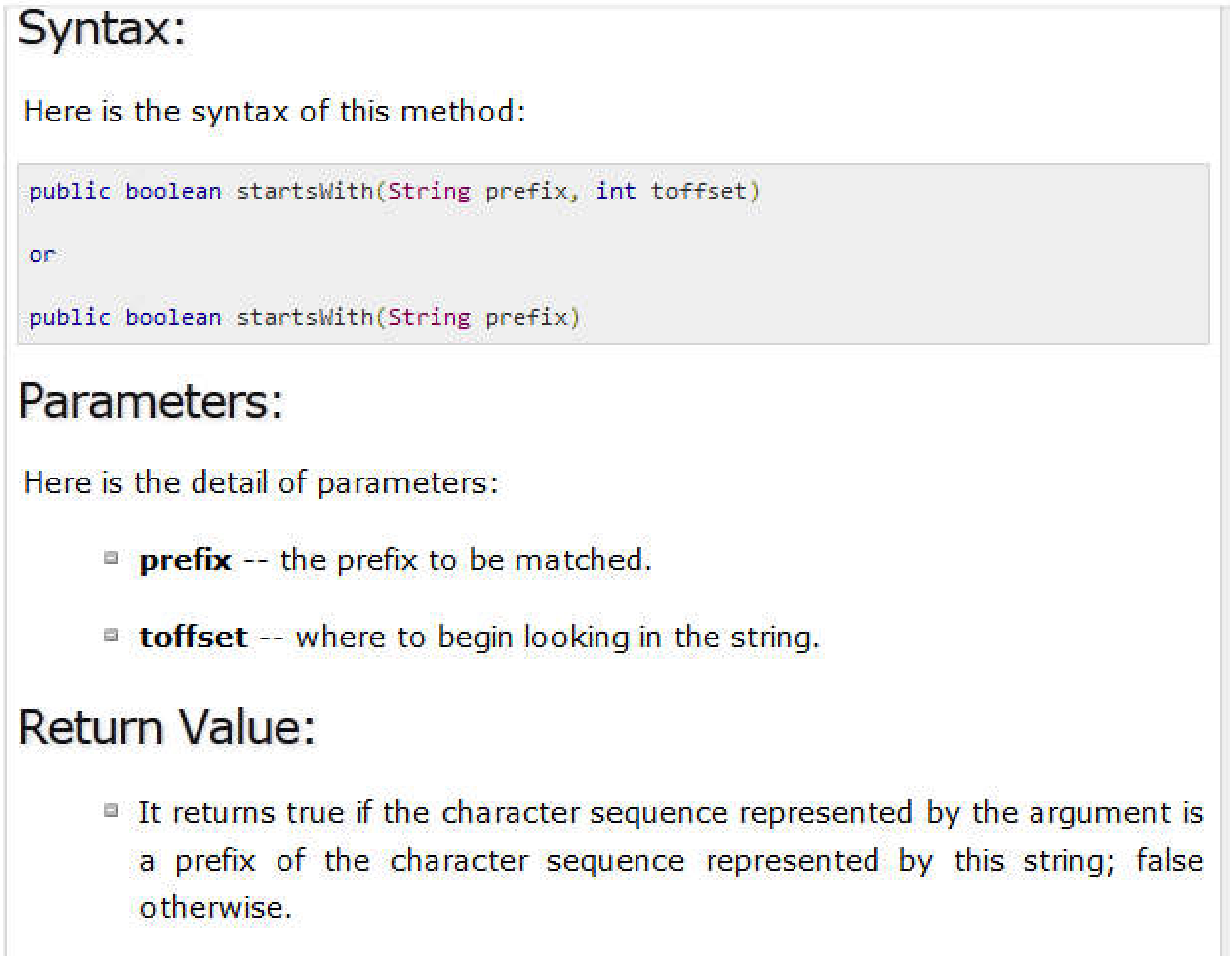
```
import java.lang.*;

public class StringDemo {

   public static void main(String[] args) {

   String str = "www.tutorialspoint.com";
   System.out.println(str);

   // the end string to be checked
   String endstr1 = ".com";
   String endstr2 = ".org";

   // checks that string str ends with given substring
   boolean retval1 = str.endsWith(endstr1);
   boolean retval2 = str.endsWith(endstr2);

   // prints true if the string ends with given substring
   System.out.println("ends with " + endstr1 + " ? " + retval1);
   System.out.println("ends with " + endstr2 + " ? " + retval2);
   }
}
```

Let us compile and run the above program, this will produce the following result:

```
www.tutorialspoint.com
ends with .com ? true
ends with .org ? false
```

Informativne metode

- **`boolean startsWith(String pref):`** proverava da li tekući string počinje stringom pref



Syntax:

Here is the syntax of this method:

```
public boolean startsWith(String prefix, int toffset)

or

public boolean startsWith(String prefix)
```

Parameters:

Here is the detail of parameters:

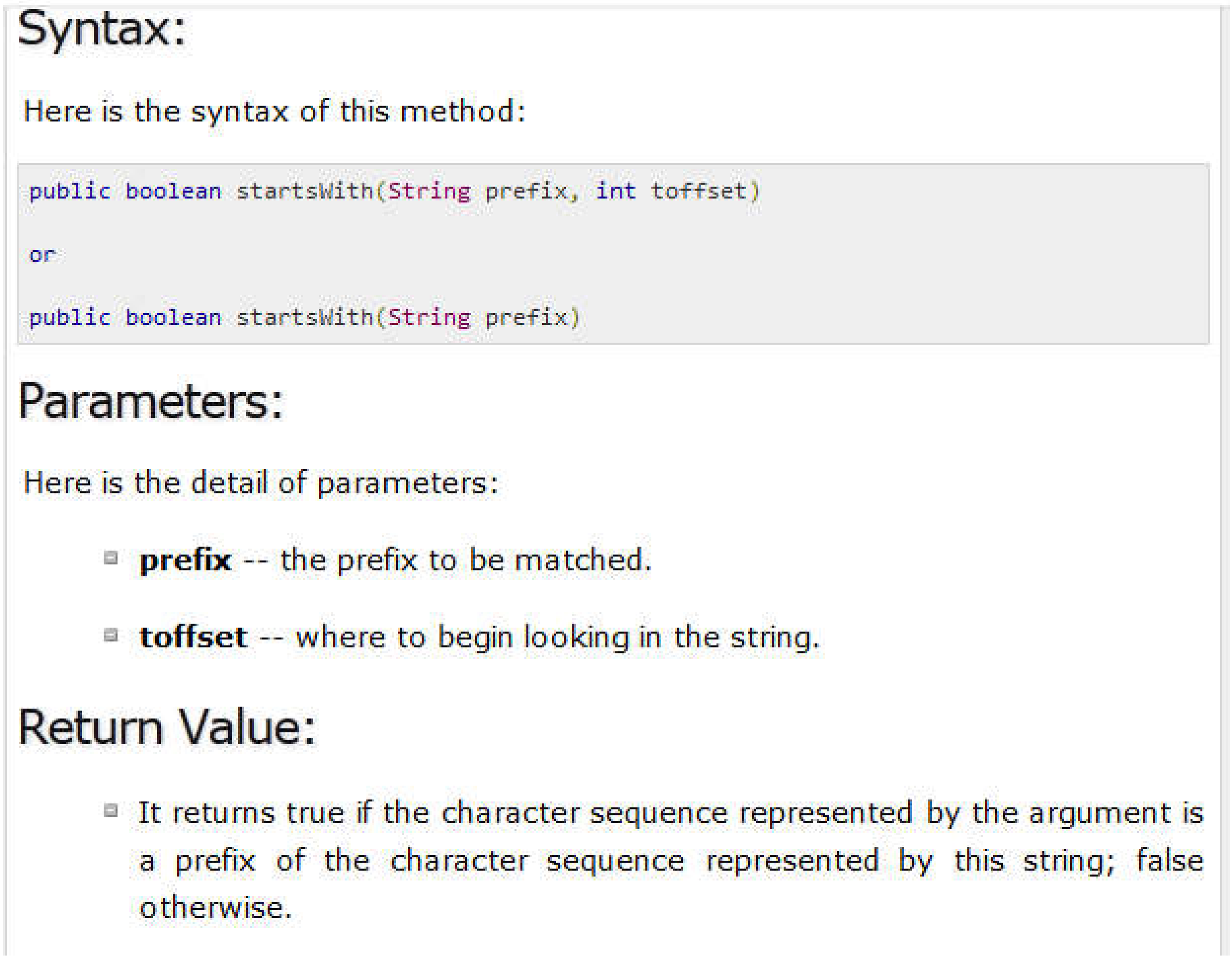
- **prefix** -- the prefix to be matched.
- **toffset** -- where to begin looking in the string.

Return Value:

- It returns true if the character sequence represented by the argument is a prefix of the character sequence represented by this string; false otherwise.

Informativne metode

- **boolean startsWith(String pref):** proverava da li tekući string počinje stringom pref



Syntax:

Here is the syntax of this method:

```
public boolean startsWith(String prefix, int toffset)

or

public boolean startsWith(String prefix)
```

Parameters:

Here is the detail of parameters:

- **prefix** -- the prefix to be matched.
- **toffset** -- where to begin looking in the string.

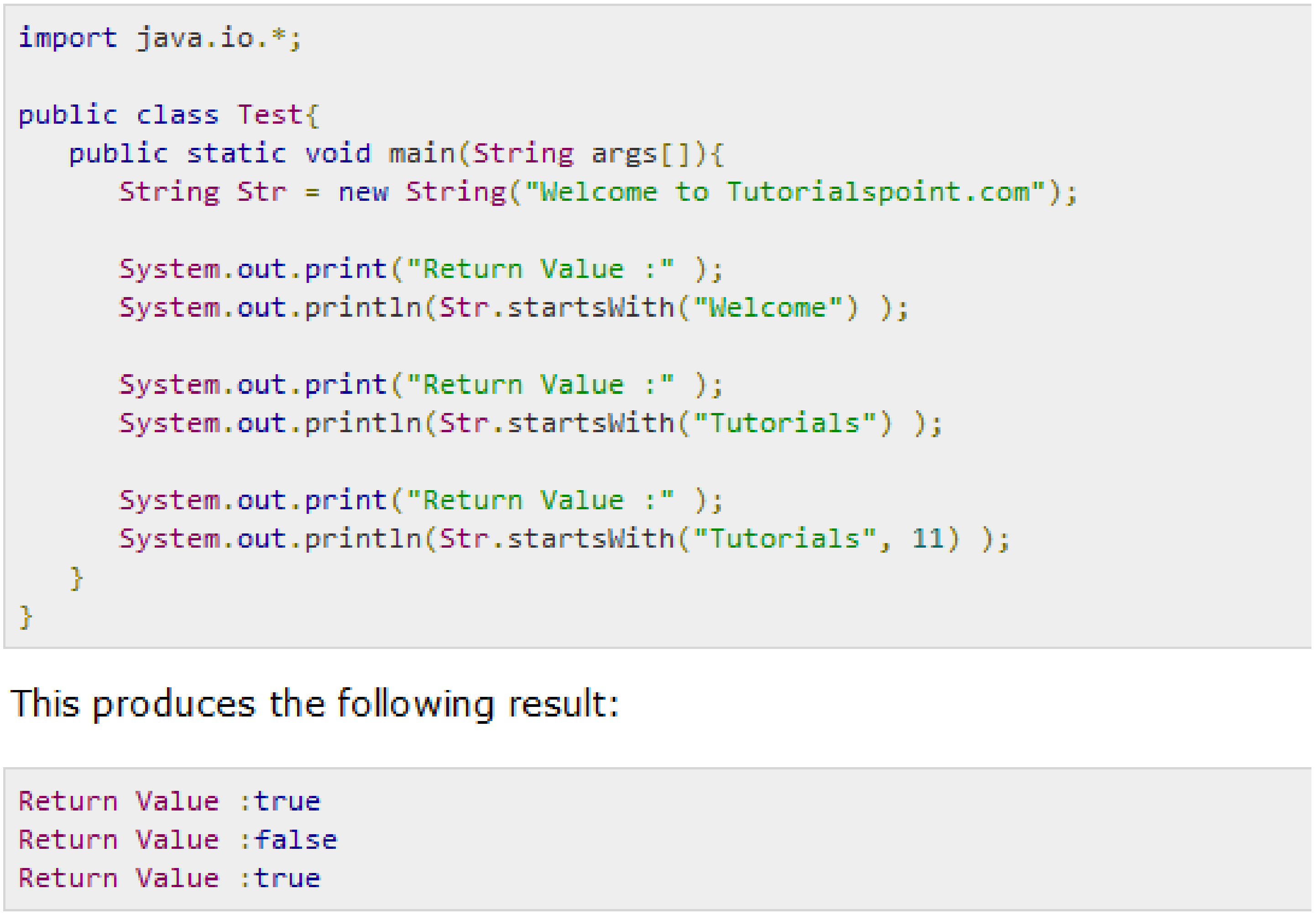
Return Value:

- It returns true if the character sequence represented by the argument is a prefix of the character sequence represented by this string; false otherwise.

Stringovi

Informativne metode

- **boolean startsWith(String pref):** proverava da li tekući string počinje stringom pref



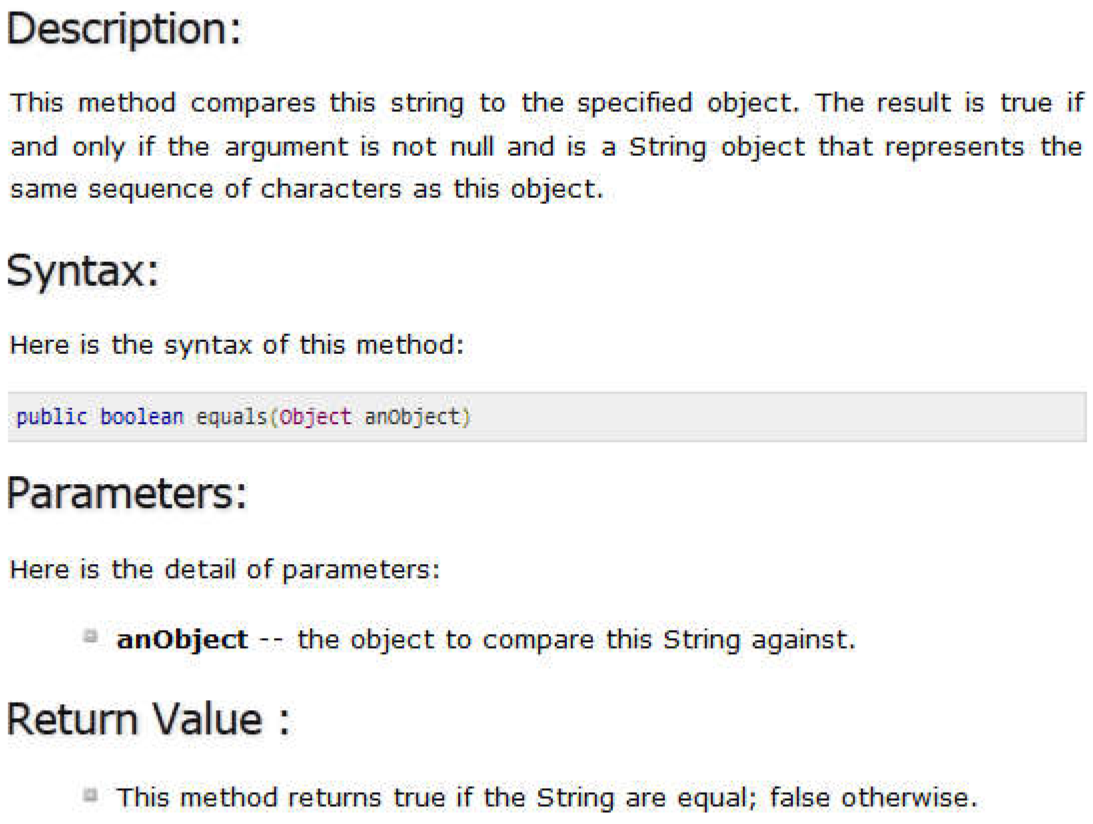
```
import java.io.*;

public class Test{
   public static void main(String args[]){
      String Str = new String("Welcome to Tutorialspoint.com");

      System.out.print("Return Value :" );
      System.out.println(Str.startsWith("Welcome") );

      System.out.print("Return Value :" );
      System.out.println(Str.startsWith("Tutorials") );

      System.out.print("Return Value :" );
      System.out.println(Str.startsWith("Tutorials", 11) );
   }
}
```

This produces the following result:

```
Return Value :true
Return Value :false
Return Value :true
```

Informativne metode

- **Boolean equals(Object anObject):** proverava da li je sadržaj tekućeg stringa jednak sadržaju datog objekta



Description:

This method compares this string to the specified object. The result is true if and only if the argument is not null and is a String object that represents the same sequence of characters as this object.

Syntax:

Here is the syntax of this method:

```
public boolean equals(Object anObject)
```

Parameters:

Here is the detail of parameters:

- **anObject** -- the object to compare this String against.

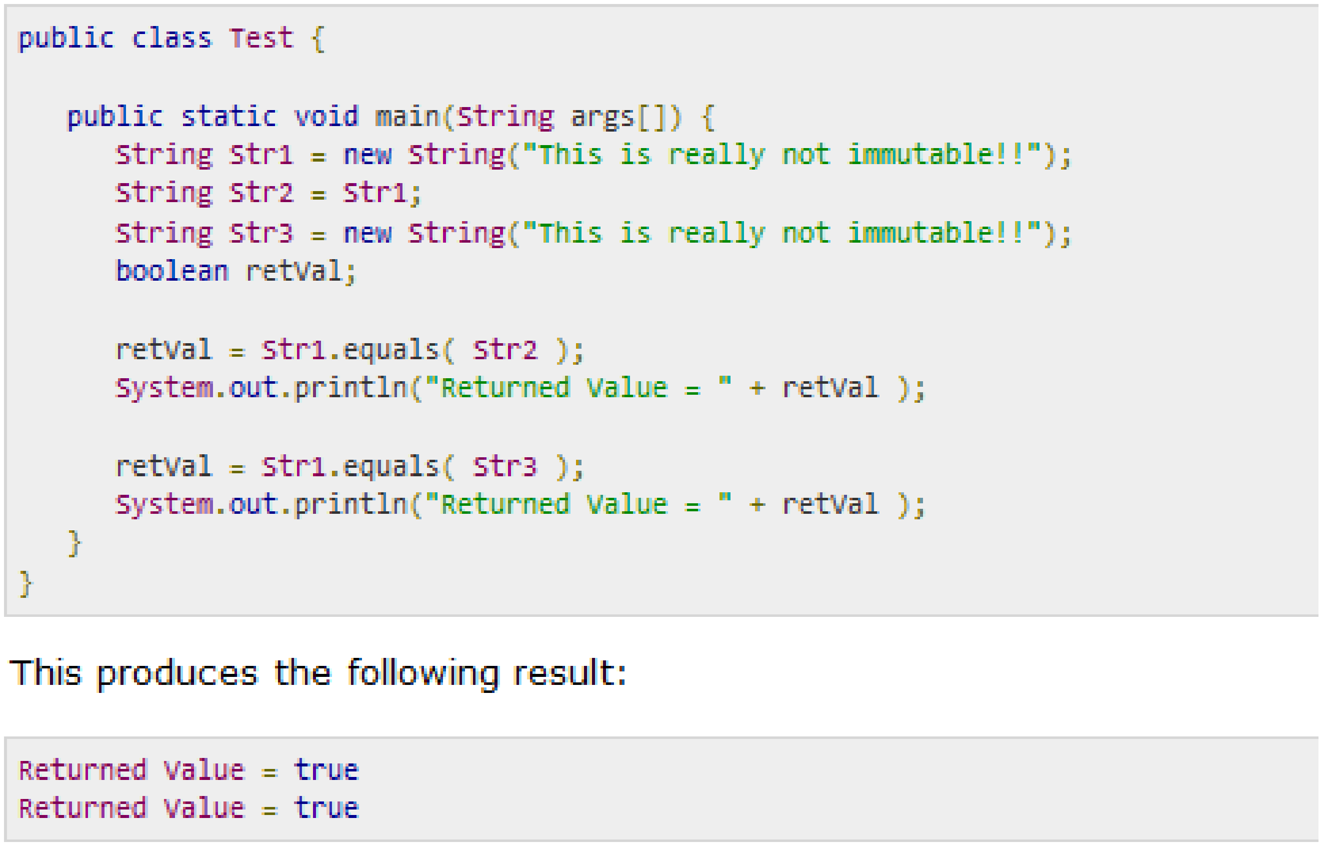
Return Value :

- This method returns true if the String are equal; false otherwise.

Stringovi

Informativne metode

- **Boolean equals(Object anObject):** proverava da li je sadržaj tekućeg stringa jednak sadržaju datog objekta



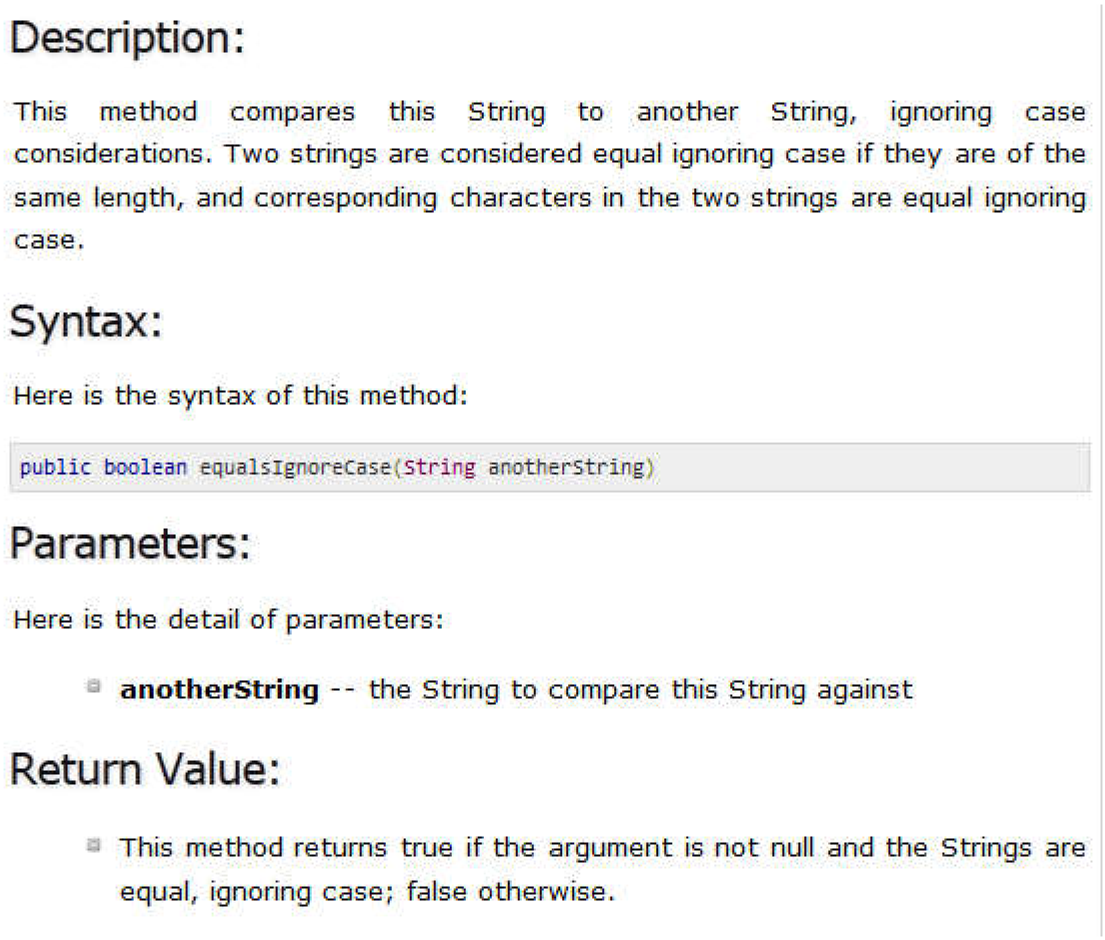
```
public class Test {

   public static void main(String args[]) {
      String Str1 = new String("This is really not immutable!!");
      String Str2 = Str1;
      String Str3 = new String("This is really not immutable!!");
      boolean retVal;

      retVal = Str1.equals( Str2 );
      System.out.println("Returned Value = " + retVal );

      retVal = Str1.equals( Str3 );
      System.out.println("Returned Value = " + retVal );
   }
}
```

This produces the following result:

```
Returned Value = true
Returned Value = true
```

Informativne metode

- **Boolean equalsIgnoreCase(String anotherString):** proverava da li je sadržaj tekućeg stringa jednak sadržaju datog stringa, ignorišući razliku između velikih i malih slova



Description:

This method compares this String to another String, ignoring case considerations. Two strings are considered equal ignoring case if they are of the same length, and corresponding characters in the two strings are equal ignoring case.

Syntax:

Here is the syntax of this method:

```
public boolean equalsIgnoreCase(String anotherString)
```

Parameters:

Here is the detail of parameters:

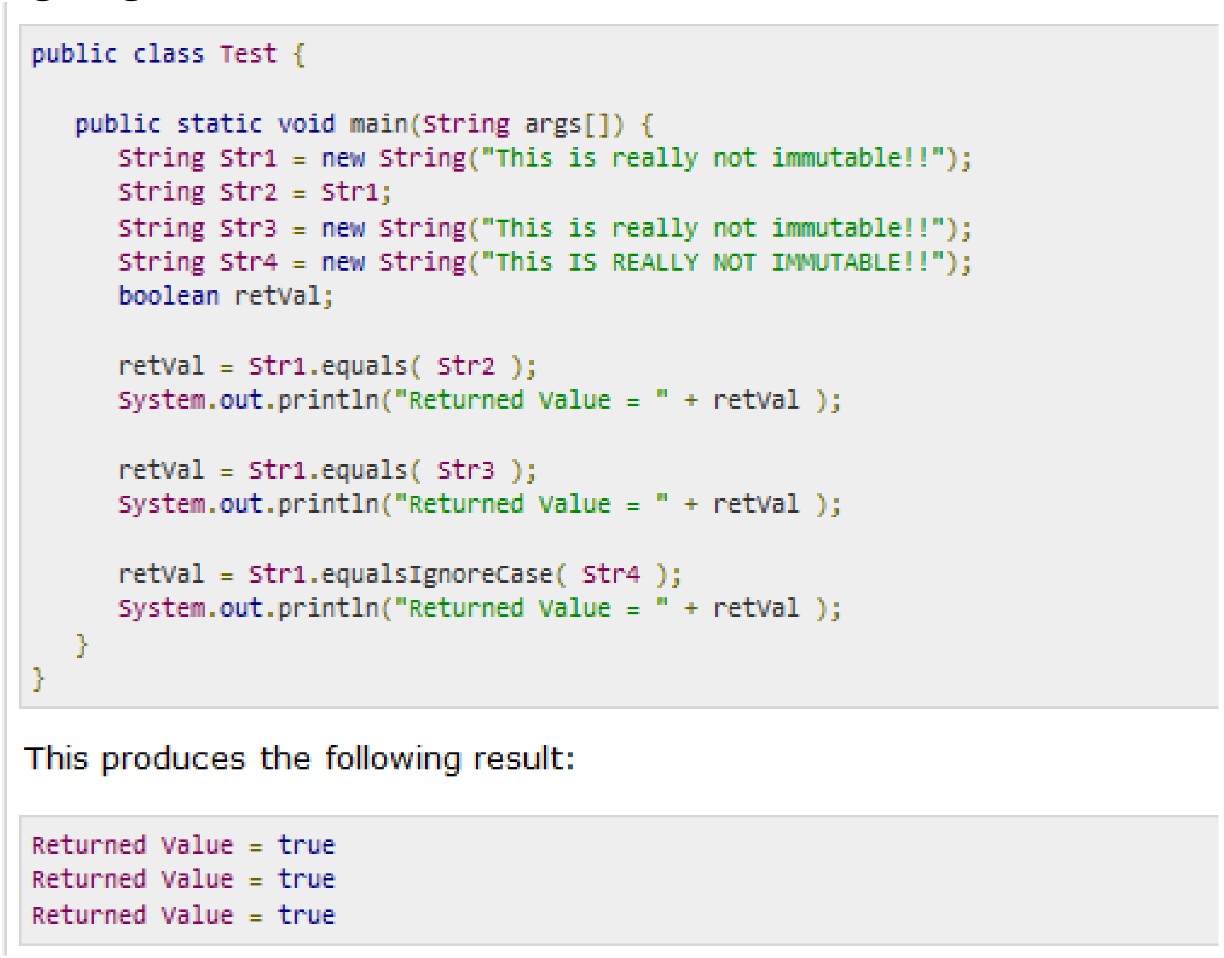
- **anotherString** -- the String to compare this String against

Return Value:

- This method returns true if the argument is not null and the Strings are equal, ignoring case; false otherwise.

Informativne metode

- **Boolean equalsIgnoreCase(String anotherString):** proverava da li je sadržaj tekućeg stringa jednak sadržaju datog stringa, ignorišući razliku između velikih i malih slova



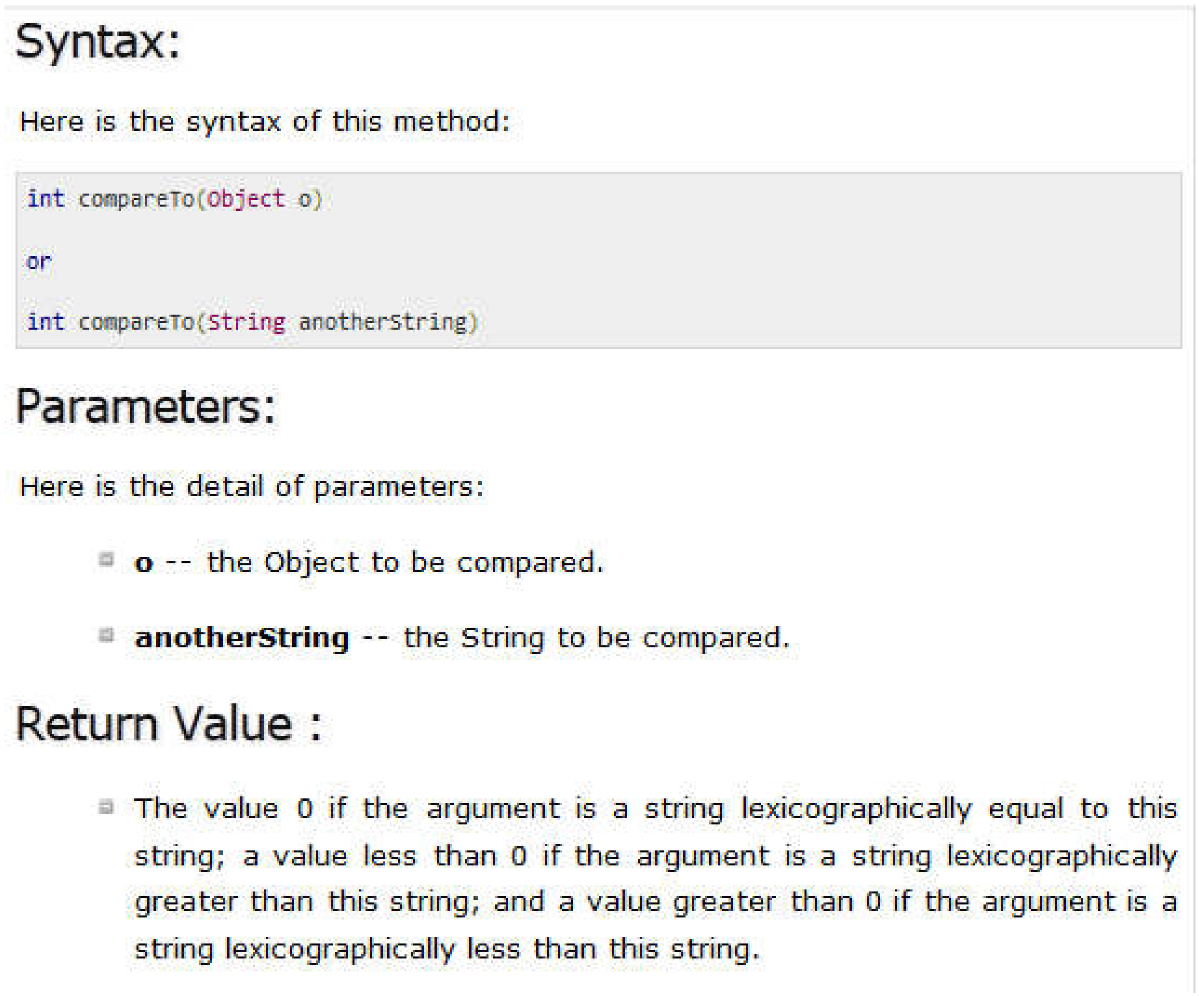
```
public class Test {

   public static void main(String args[]) {
      String Str1 = new String("This is really not immutable!!");
      String Str2 = Str1;
      String Str3 = new String("This is really not immutable!!");
      String Str4 = new String("This IS REALLY NOT IMMUTABLE!!");
      boolean retVal;

      retVal = Str1.equals( Str2 );
      System.out.println("Returned Value = " + retVal );

      retVal = Str1.equals( Str3 );
      System.out.println("Returned Value = " + retVal );

      retVal = Str1.equalsIgnoreCase( Str4 );
      System.out.println("Returned Value = " + retVal );
   }
}
```

This produces the following result:

```
Returned Value = true
Returned Value = true
Returned Value = true
```

Informativne metode

- **`Int compareTo(String anotherString):`** leksikografsko poređenje stringova



Syntax:

Here is the syntax of this method:

```
int compareTo(Object o)

or

int compareTo(String anotherString)
```

Parameters:

Here is the detail of parameters:

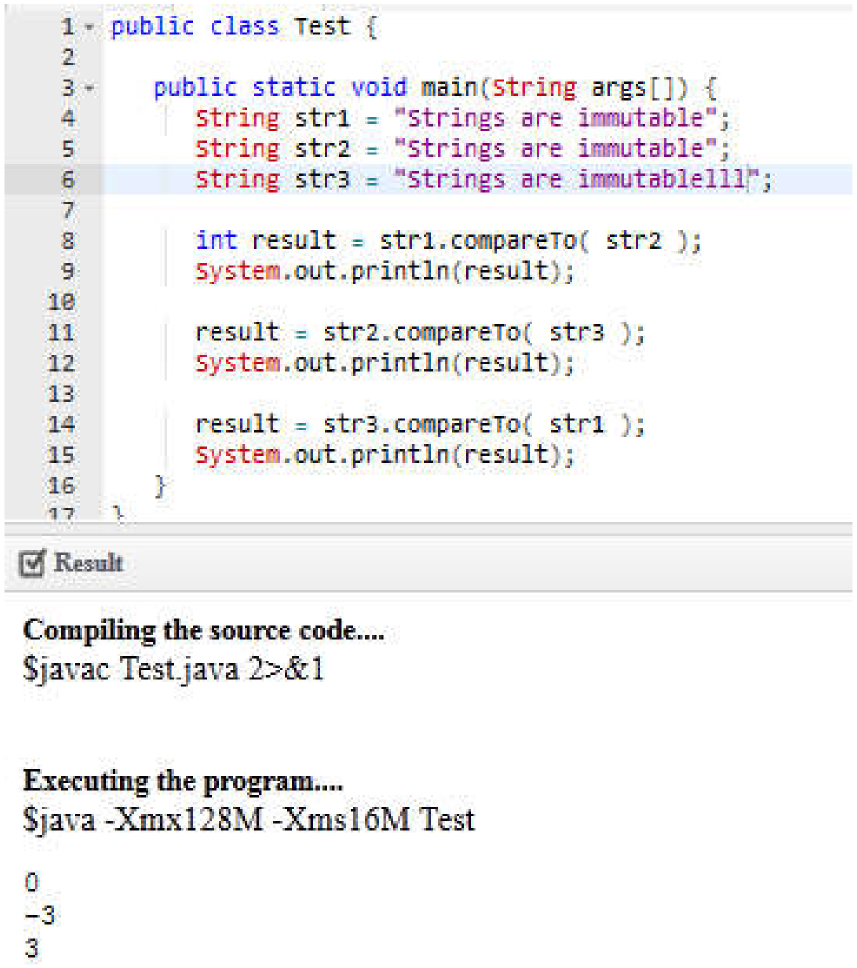
- **o** -- the Object to be compared.
- **anotherString** -- the String to be compared.

Return Value :

- The value 0 if the argument is a string lexicographically equal to this string; a value less than 0 if the argument is a string lexicographically greater than this string; and a value greater than 0 if the argument is a string lexicographically less than this string.

Informativne metode

- **`Int compareTo(String anotherString):`** leksikografsko poređenje stringova



```
public class Test {

    public static void main(String args[]) {
        String str1 = "Strings are immutable";
        String str2 = "Strings are immutable";
        String str3 = "Strings are immutablelll";

        int result = str1.compareTo( str2 );
        System.out.println(result);

        result = str2.compareTo( str3 );
        System.out.println(result);

        result = str3.compareTo( str1 );
        System.out.println(result);
    }
}
```

Result

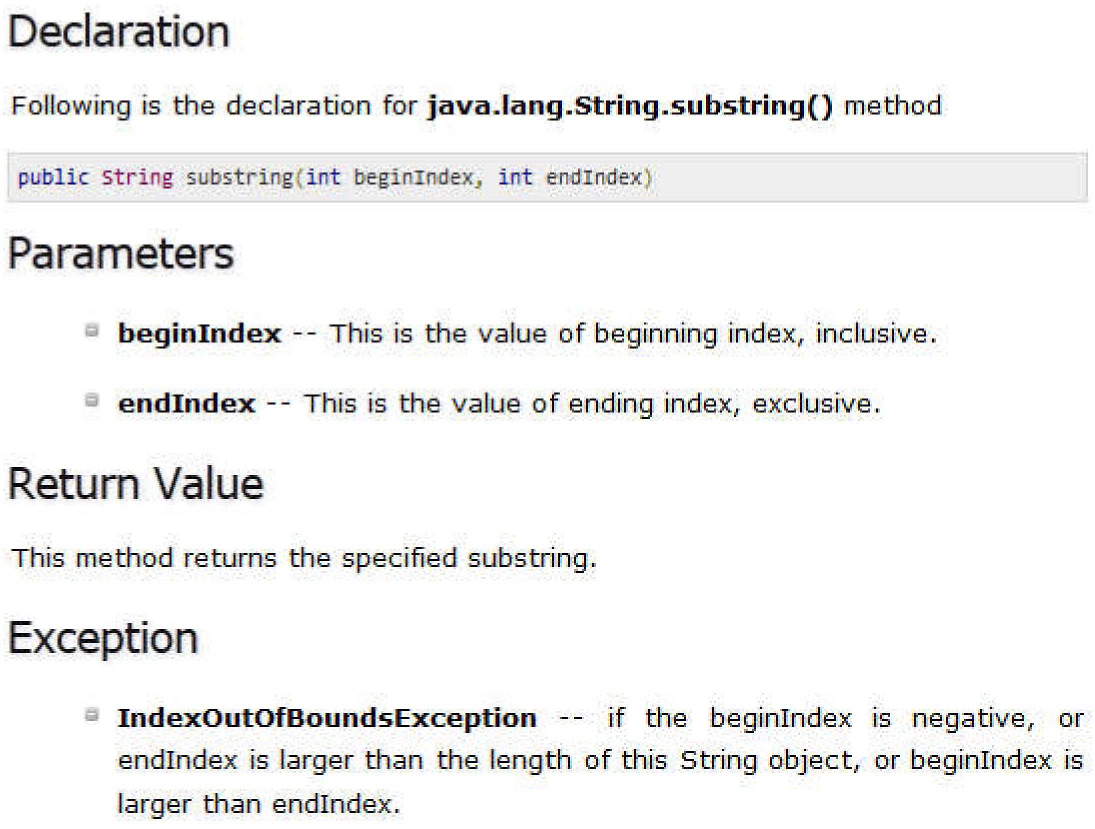
```
Compiling the source code....
$javac Test.java 2>&1


Executing the program....
$java -Xmx128M -Xms16M Test

0
-3
3
```

Transformativne metode

- Ova vrsta metoda pravi nove instance tipa `String` iz originalnog stringa.
- **Nekoliko značajnijih jednostavnijih metoda:**
- `String substring(int beginIndex, int endIndex):` vraća podstring tekućeg stringa koji sadrži kopije karaktera originalnog stringa od pozicije `beginIndex` do pozicije `endIndex-1`,
- `String trim():` odstranjuje sve znakove koji predstavljaju beline (engl. *Whitespace*) sa obe strane tekućeg stringa,
- `String concat(String str):` spajanje stringova,
- `String ToLowerCase():` vraća string dobijen iz tekućeg stringa konvertovanjem velikih slova u mala,
- `String ToUpperCase():` vraća string dobijen iz tekućeg stringa konvertovanjem malih slova u velika.

Transformativne metode

- `String substring(int beginIndex, int endIndex):` vraća podstring tekućeg stringa koji sadrži kopije karaktera originalnog stringa od pozicije `beginIndex` do pozicije `endIndex-1`,



Declaration

Following is the declaration for **java.lang.String.substring()** method

```
public String substring(int beginIndex, int endIndex)
```

Parameters

- **beginIndex** -- This is the value of beginning index, inclusive.
- **endIndex** -- This is the value of ending index, exclusive.

Return Value

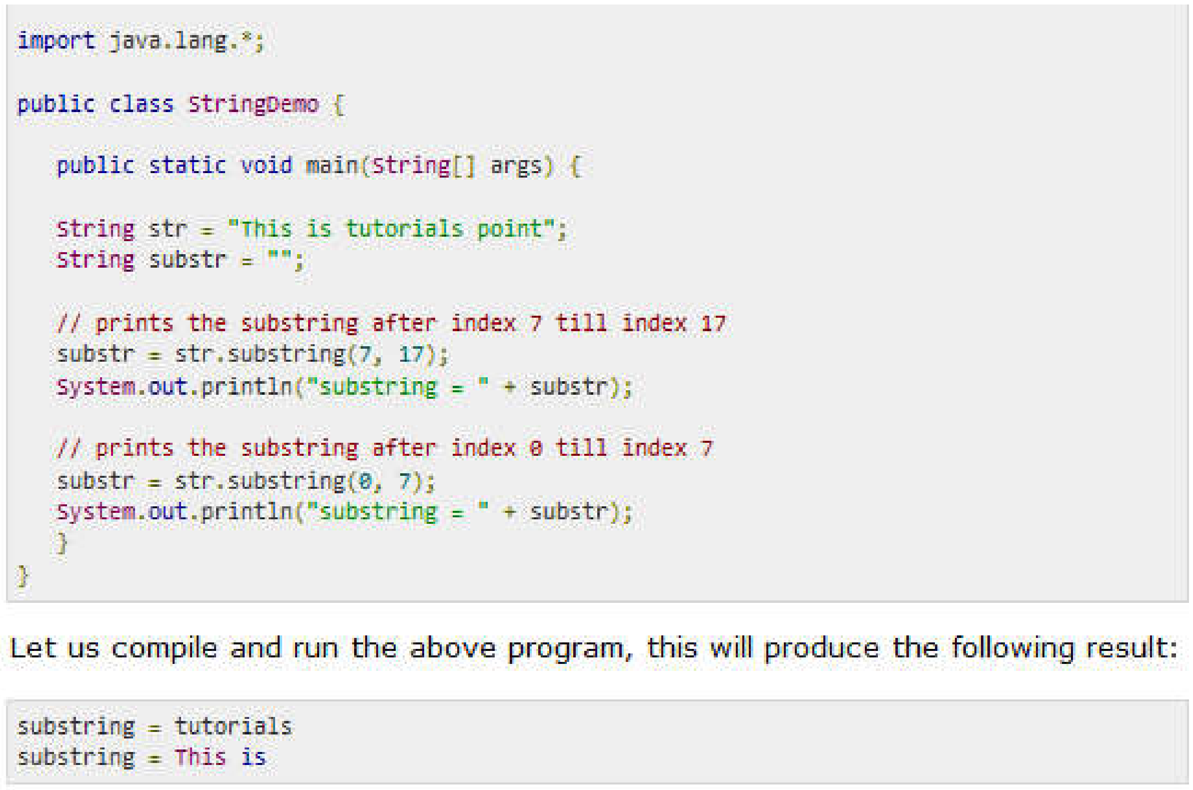
This method returns the specified substring.

Exception

- **IndexOutOfBoundsException** -- if the beginIndex is negative, or endIndex is larger than the length of this String object, or beginIndex is larger than endIndex.

Transformativne metode

- **String substring(int beginIndex, int endIndex):** vraća podstring tekućeg stringa koji sadrži kopije karaktera originalnog stringa od pozicije **beginIndex** do pozicije **endIndex-1**,



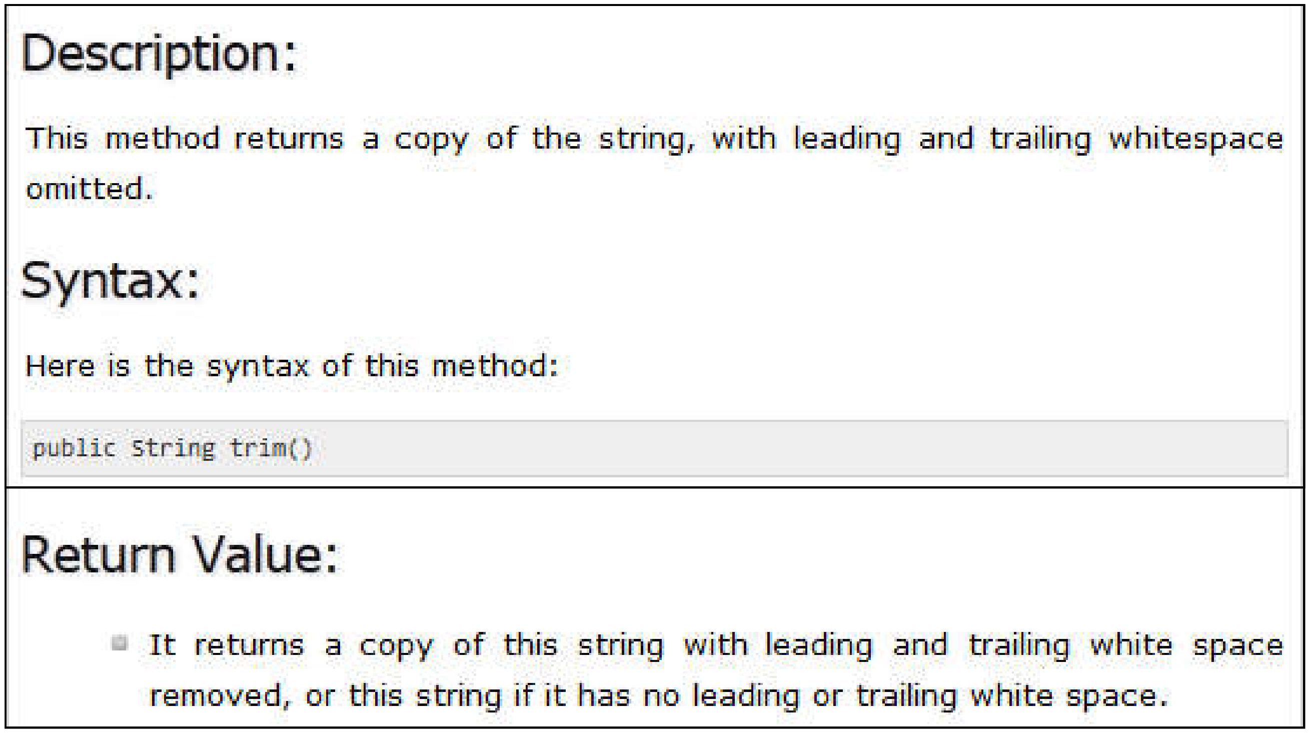
```
import java.lang.*;

public class StringDemo {

   public static void main(String[] args) {

   String str = "This is tutorials point";
   String substr = "";

   // prints the substring after index 7 till index 17
   substr = str.substring(7, 17);
   System.out.println("substring = " + substr);

   // prints the substring after index 0 till index 7
   substr = str.substring(0, 7);
   System.out.println("substring = " + substr);
   }
}
```

Let us compile and run the above program, this will produce the following result:

```
substring = tutorials
substring = This is
```

Transformativne metode

- **`String trim():`** odstranjuje sve znakove koji predstavljaju beline (engl. *Whitespace*) sa obe strane tekućeg stringa



Description:

This method returns a copy of the string, with leading and trailing whitespace omitted.

Syntax:

Here is the syntax of this method:

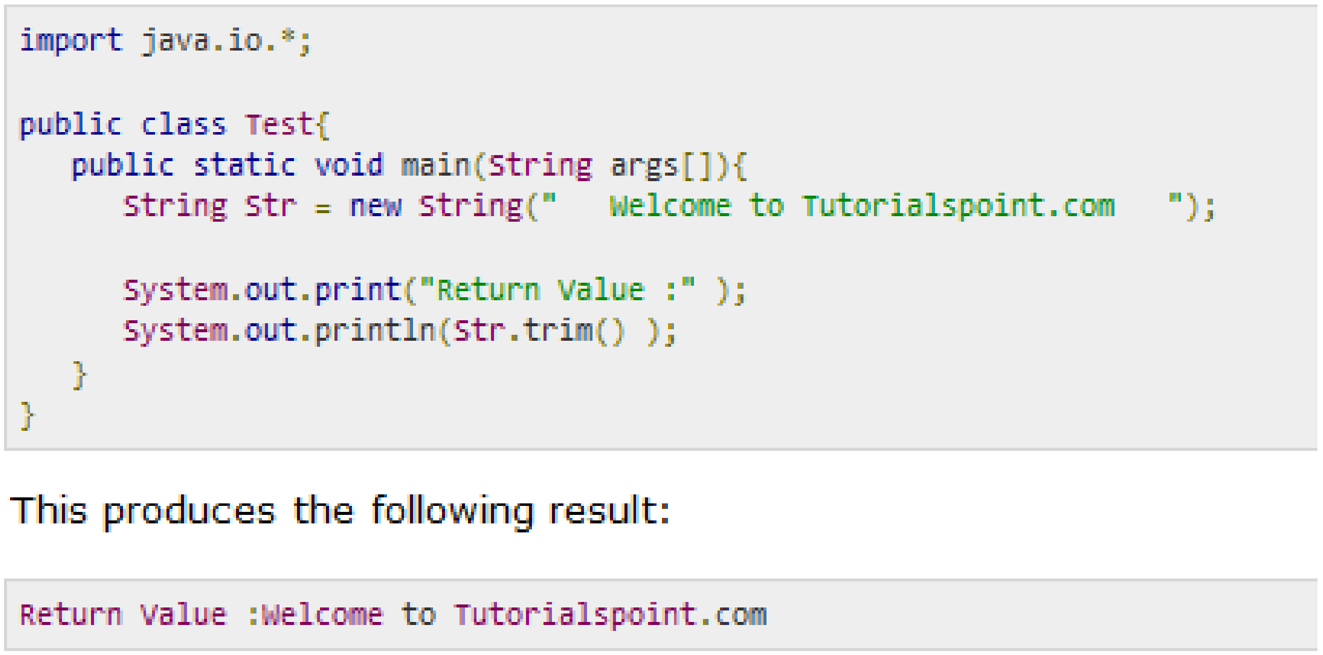
```
public String trim()
```

Return Value:

- It returns a copy of this string with leading and trailing white space removed, or this string if it has no leading or trailing white space.

Transformativne metode

- **`String trim():`** odstranjuje sve znakove koji predstavljaju beline (engl. *Whitespace*) sa obe strane tekućeg stringa



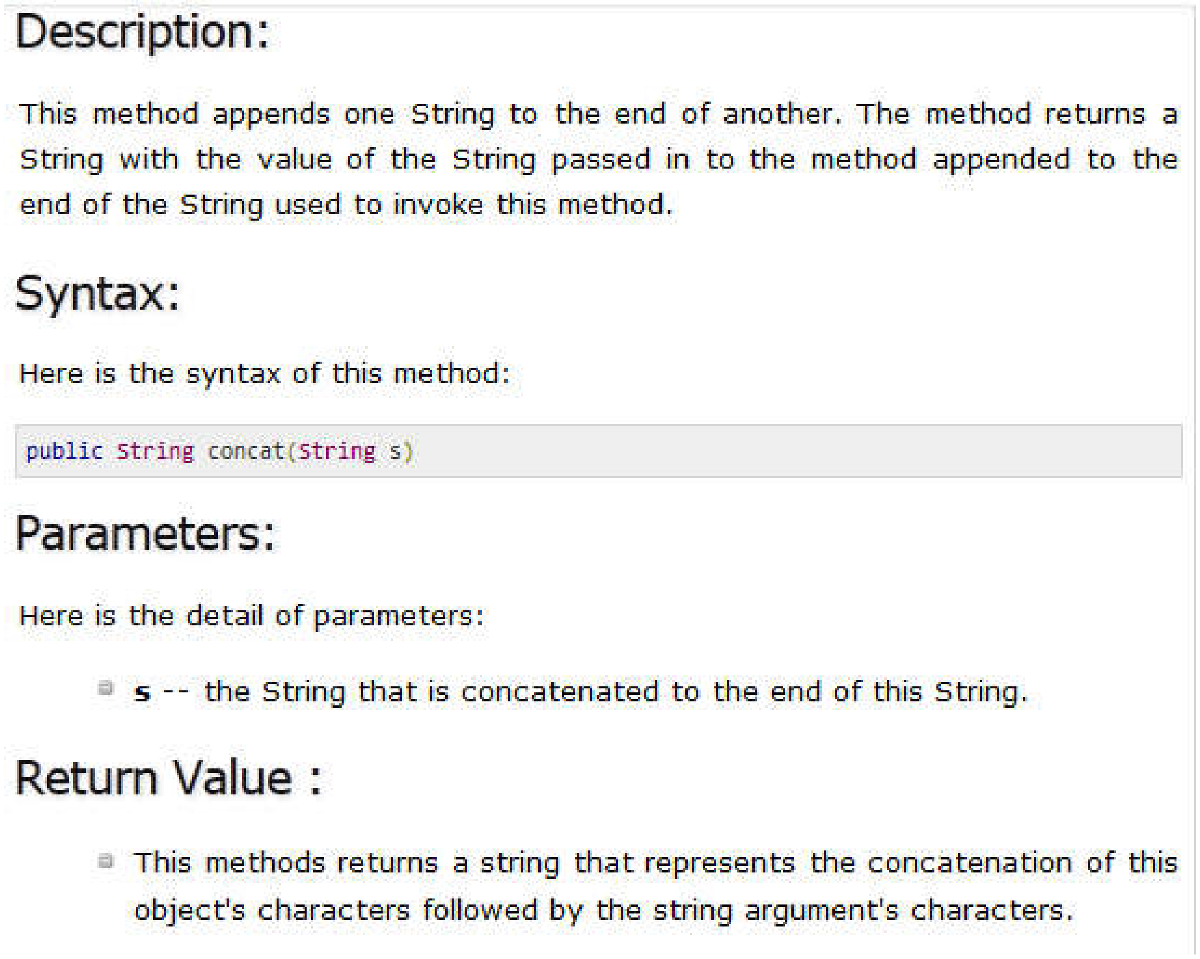
```
import java.io.*;

public class Test{
   public static void main(String args[]){
      String Str = new String("   Welcome to Tutorialspoint.com   ");

      System.out.print("Return Value :" );
      System.out.println(Str.trim() );
   }
}
```

This produces the following result:

```
Return Value :Welcome to Tutorialspoint.com
```

Transformativne metode

- **`String concat(String str):`** spajanje stringova



Description:

This method appends one String to the end of another. The method returns a String with the value of the String passed in to the method appended to the end of the String used to invoke this method.

Syntax:

Here is the syntax of this method:

```
public String concat(String s)
```

Parameters:

Here is the detail of parameters:

- **s** -- the String that is concatenated to the end of this String.

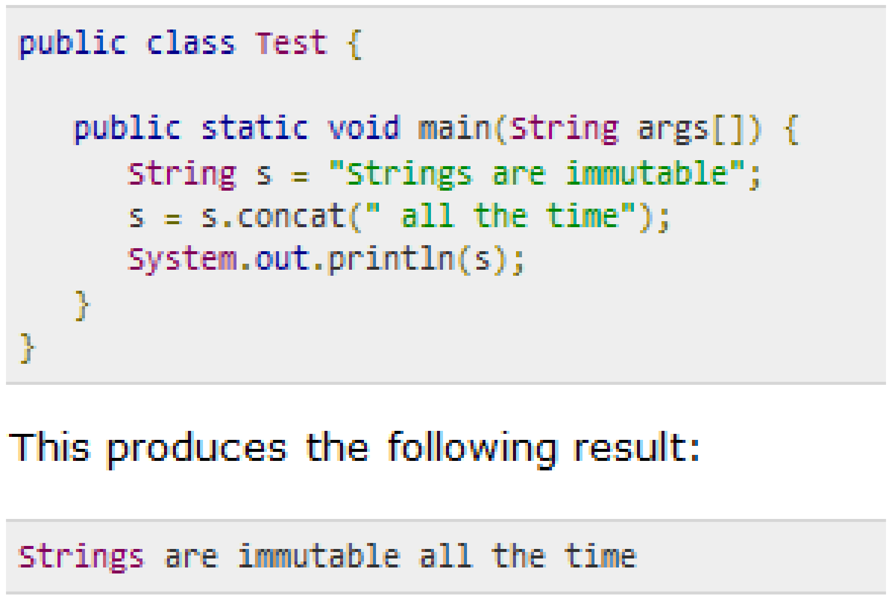
Return Value :

- This methods returns a string that represents the concatenation of this object's characters followed by the string argument's characters.

Stringovi

Transformativne metode

- **`String concat(String str):`** spajanje stringova



```
public class Test {

   public static void main(String args[]) {
      String s = "Strings are immutable";
      s = s.concat(" all the time");
      System.out.println(s);
   }
}
```

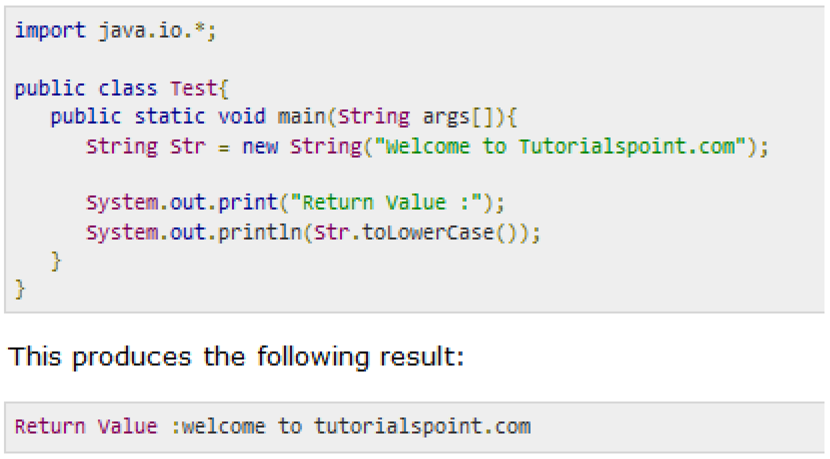
This produces the following result:

```
Strings are immutable all the time
```

Stringovi

Transformativne metode

- **String ToLowerCase():** vraća string dobijen iz tekućeg stringa konvertovanjem velikih slova u mala

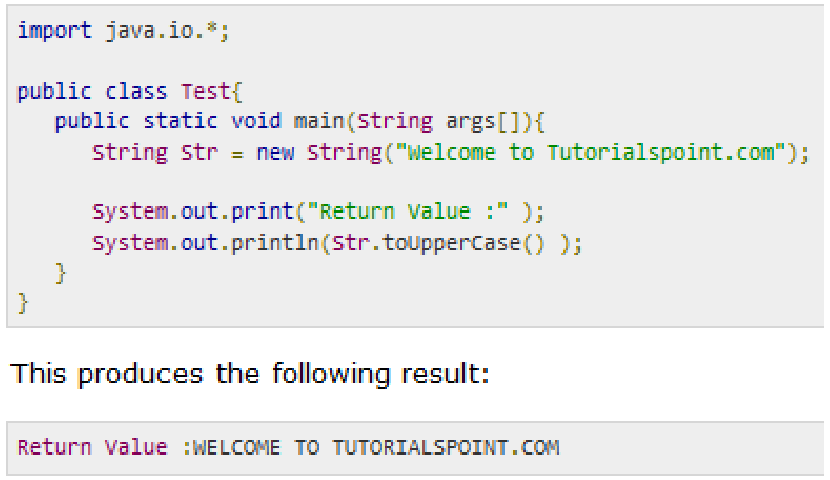


```
import java.io.*;

public class Test{
   public static void main(String args[]){
      String Str = new String("Welcome to Tutorialspoint.com");

      System.out.print("Return Value :");
      System.out.println(Str.toLowerCase());
   }
}
```

This produces the following result:

```
Return Value :welcome to tutorialspoint.com
```

- **String ToUpperCase():** vraća string dobijen iz tekućeg stringa konvertovanjem malih slova u velika.



```
import java.io.*;

public class Test{
   public static void main(String args[]){
      String Str = new String("Welcome to Tutorialspoint.com");

      System.out.print("Return Value :" );
      System.out.println(Str.toUpperCase() );
   }
}
```

This produces the following result:

```
Return Value :WELCOME TO TUTORIALSPOINT.COM
```

Transformativne metode

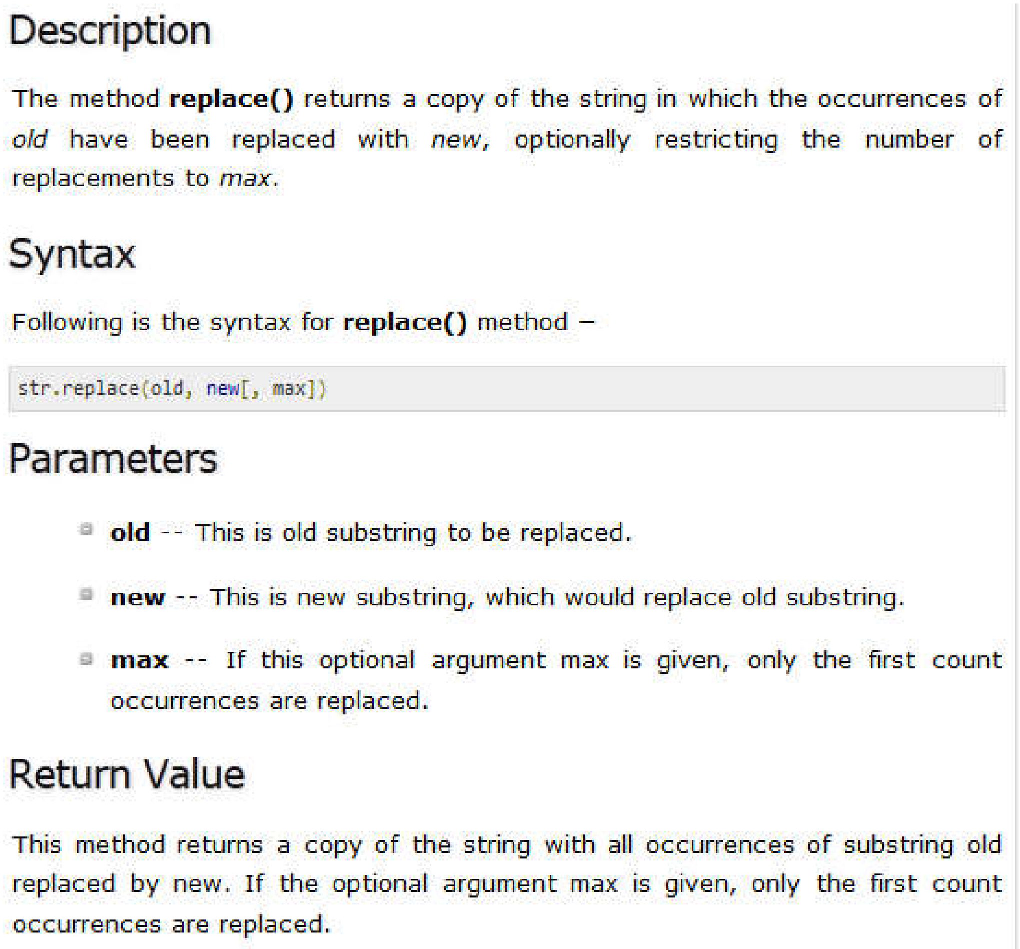
Nekoliko naprednijih metoda:

- **`String replace(CharSequence target, CharSequence replacement)`**: zamenjuje svaku pojavu podstringa `target` u tekućem stringu sa `replacement`,
- **`String replaceFirst(String regex, String replacement)`** zamenjuje prvu pojavu podstringa koji odgovara regularnom izrazu `regex` sa `replacement`,
- **`String replaceAll(String regex, String replacement)`** zamenjuje sve pojave podstringa koji odgovara regularnom izrazu `regex` sa `replacement`,
- **`String[] split(String regex)`**: deli string u podstringove koristeći dati regularan izraz kao separator.

Transformativne metode

Nekoliko naprednijih metoda:

- **`String replace(CharSequence target, CharSequence replacement)`** : zamenjuje svaku pojavu podstringa `target` u tekućem stringu sa `replacement`,



Description

The method **replace()** returns a copy of the string in which the occurrences of *old* have been replaced with *new*, optionally restricting the number of replacements to *max*.

Syntax

Following is the syntax for **replace()** method −

```
str.replace(old, new[, max])
```

Parameters

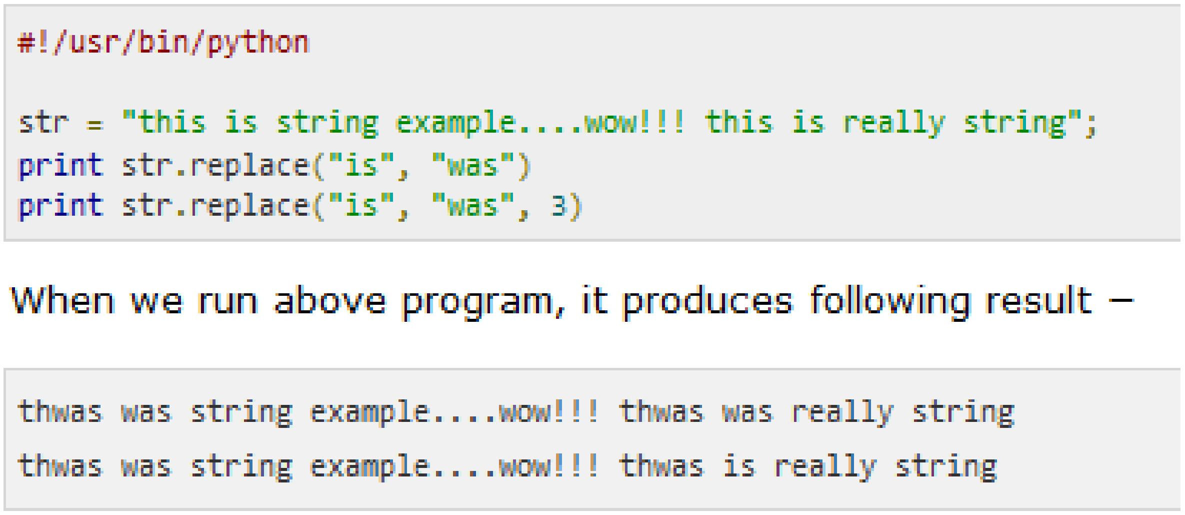
- **old** -- This is old substring to be replaced.
- **new** -- This is new substring, which would replace old substring.
- **max** -- If this optional argument max is given, only the first count occurrences are replaced.

Return Value

This method returns a copy of the string with all occurrences of substring old replaced by new. If the optional argument max is given, only the first count occurrences are replaced.

Transformativne metode

- **`String replace(CharSequence target, CharSequence replacement)`** : zamenjuje svaku pojavu podstringa `target` u tekućem stringu sa `replacement`



```
#!/usr/bin/python

str = "this is string example....wow!!! this is really string";
print str.replace("is", "was")
print str.replace("is", "was", 3)
```

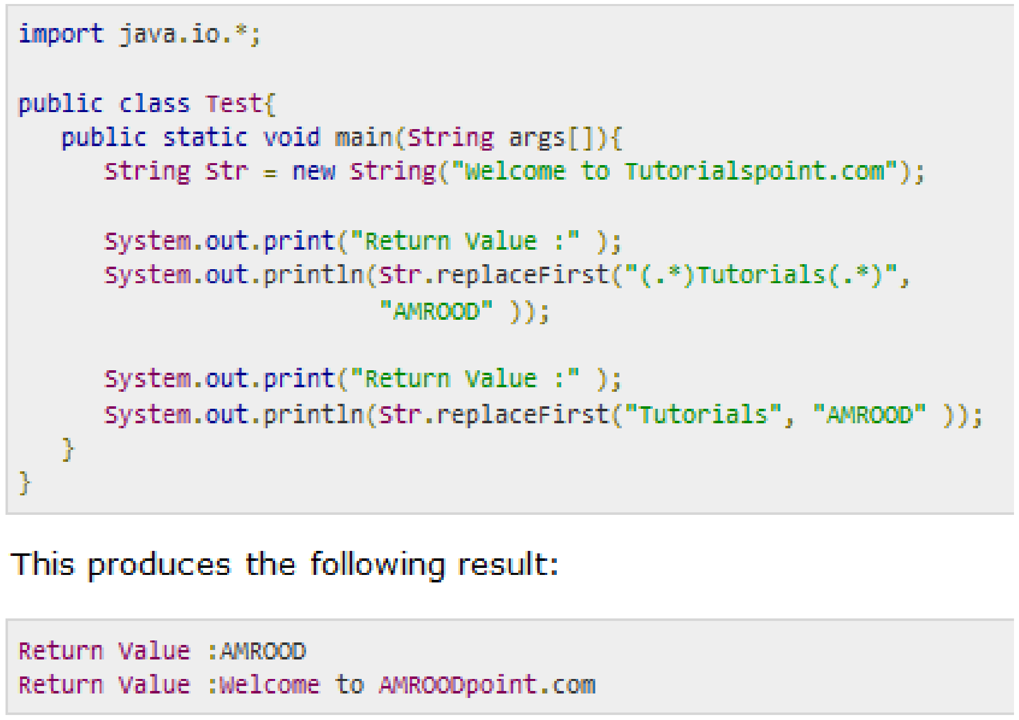
When we run above program, it produces following result –

```
thwas was string example....wow!!! thwas was really string
thwas was string example....wow!!! thwas is really string
```

Stringovi

Transformativne metode

- **`String replaceFirst(String regex, String replacement)`** zamenjuje prvu pojavu podstringa koji odgovara regularnom izrazu `regex` sa `replacement`



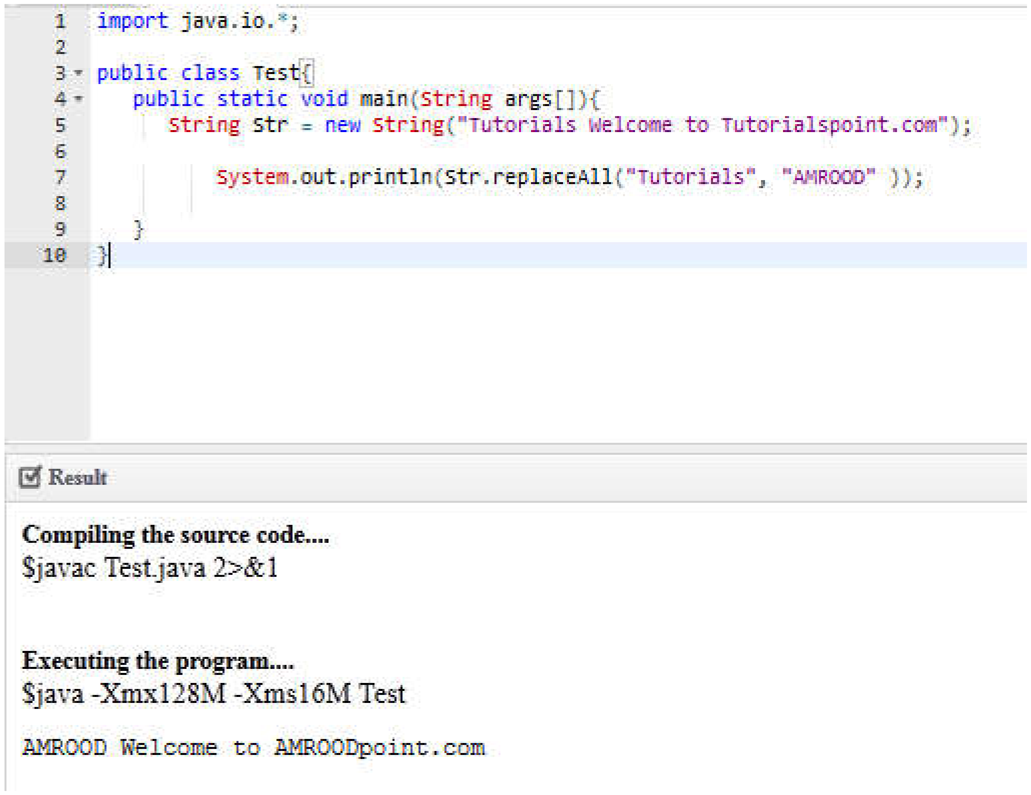
```
import java.io.*;

public class Test{
   public static void main(String args[]){
      String Str = new String("Welcome to Tutorialspoint.com");

      System.out.print("Return Value :" );
      System.out.println(Str.replaceFirst("(.*)Tutorials(.*)",
                         "AMROOD" ));

      System.out.print("Return Value :" );
      System.out.println(Str.replaceFirst("Tutorials", "AMROOD" ));
   }
}
```

This produces the following result:

```
Return Value :AMROOD
Return Value :Welcome to AMROODpoint.com
```

Transformativne metode

- **`String replaceAll(String regex, String replacement)`** zamenjuje sve pojave podstringa koji odgovaraju regularnom izrazu `regex` sa `replacement`



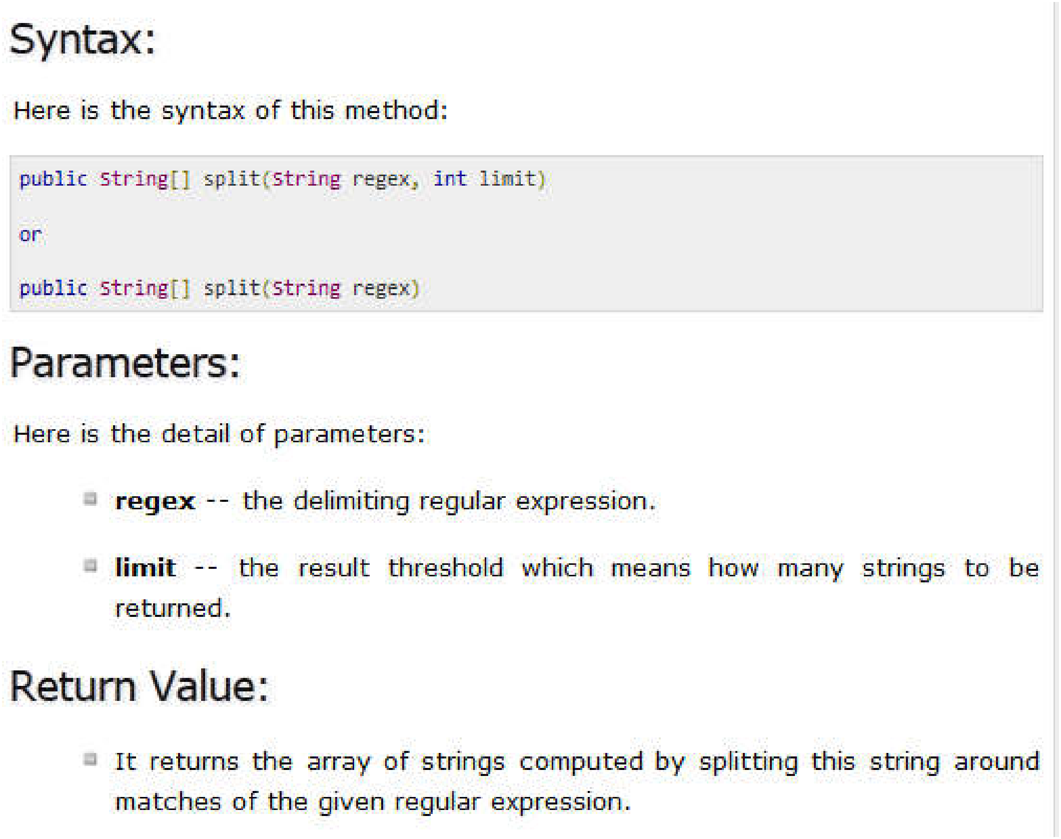
```java
import java.io.*;

public class Test{
   public static void main(String args[]){
      String Str = new String("Tutorials Welcome to Tutorialspoint.com");

          System.out.println(Str.replaceAll("Tutorials", "AMROOD" ));

   }
}
```

Result

```
Compiling the source code....
$javac Test.java 2>&1


Executing the program....
$java -Xmx128M -Xms16M Test

AMROOD Welcome to AMROODpoint.com
```

Transformativne metode

- **`String[] split(String regex):`** deli string u podstringove koristeći dati regularan izraz kao separator.



Syntax:

Here is the syntax of this method:

```
public String[] split(String regex, int limit)

or

public String[] split(String regex)
```

Parameters:

Here is the detail of parameters:

- **regex** -- the delimiting regular expression.
- **limit** -- the result threshold which means how many strings to be returned.

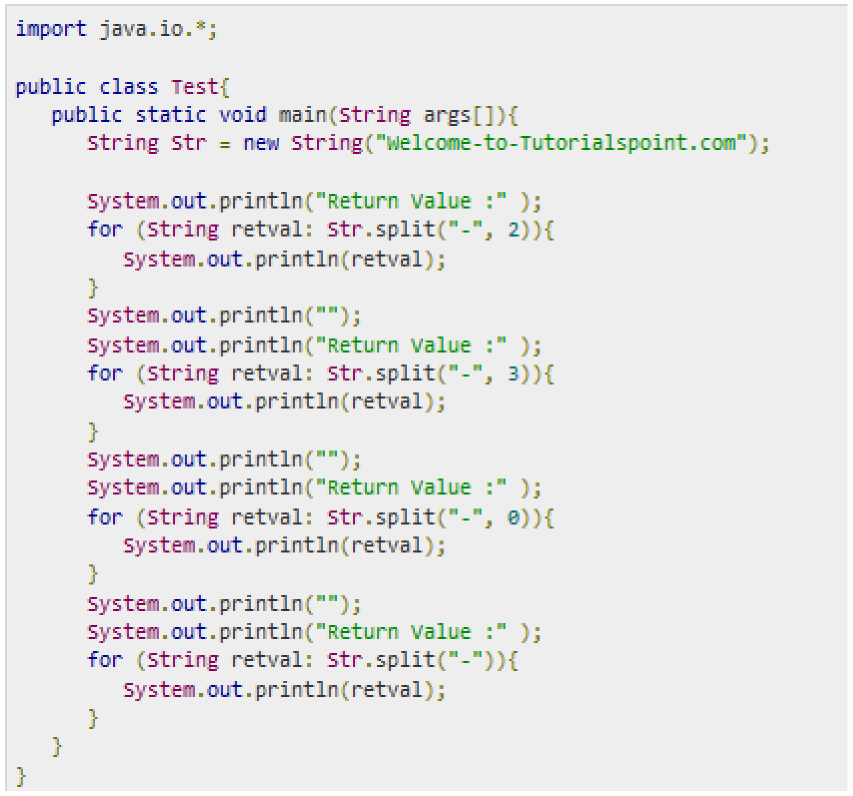
Return Value:

- It returns the array of strings computed by splitting this string around matches of the given regular expression.

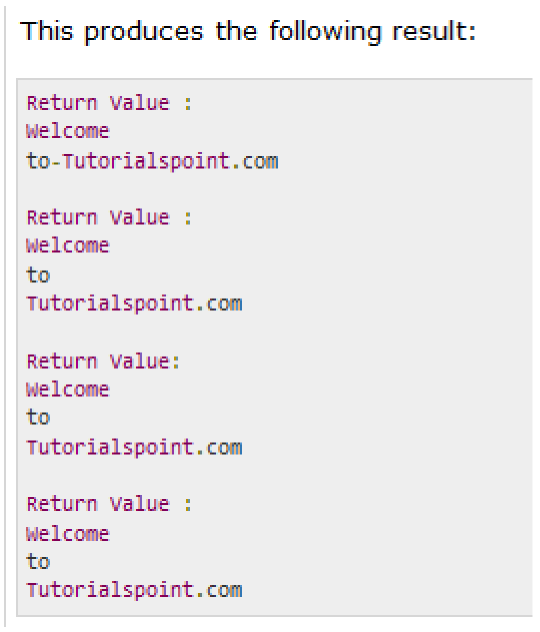
Stringovi

Transformativne metode

- **`String[] split(String regex):`** deli string u podstringove koristeći dati regularan izraz kao separator



```
import java.io.*;

public class Test{
   public static void main(String args[]){
      String Str = new String("Welcome-to-Tutorialspoint.com");

      System.out.println("Return Value :" );
      for (String retval: Str.split("-", 2)){
         System.out.println(retval);
      }
      System.out.println("");
      System.out.println("Return Value :" );
      for (String retval: Str.split("-", 3)){
         System.out.println(retval);
      }
      System.out.println("");
      System.out.println("Return Value :" );
      for (String retval: Str.split("-", 0)){
         System.out.println(retval);
      }
      System.out.println("");
      System.out.println("Return Value :" );
      for (String retval: Str.split("-")){
         System.out.println(retval);
      }
   }
}
```



This produces the following result:

```
Return Value :
Welcome
to-Tutorialspoint.com

Return Value :
Welcome
to
Tutorialspoint.com

Return Value:
Welcome
to
Tutorialspoint.com

Return Value :
Welcome
to
Tutorialspoint.com
```

Transformativne metode

Regularni izrazi

- Nekoliko prethodno navedenih metoda prima kao parametar string koji predstavlja regularan izraz.
- Regularni izrazi omogućavaju konciznu reprezentaciju više stringova slične strukture.
- Tačnije, svaki regularan izraz opisuje skup stringova koji ga zadovoljavaju.
- Sledeće konstrukcije se mogu koristiti za formulisanje regularnog izraza:
  - [abc]: jedan karakter a, b ili c,
  - [^abc]: bilo koji karakter osim a, b ili c (negacija),
  - [a-z]: jedan karakter iz opsega od a do z,

Transformativne metode

Regularni izrazi

- [a-zA-Z] unija dva opsega,
- [a-z&&[^df]: presek dva opsega (od a do z, bez d i f),
- \d: cifra ([0-9]),
- \s: karakter koji predstavlja belinu (whitespace)
- \w: karakter od koga se sastavljaju reči ([a-zA-Z_0-9])

Primer: Upotreba klase String

Primer 6.82: Upotreba klase String

```
public class StringExample {
  public static void main(String[] args) {
    String str = " 1 2.4 56  ".trim();
    System.out.println("Trim-ovan: '" + str + "'");

    // ekstrakcija sadrzaja XML taga
    String tag = "<data>hello, world</data>";
    int poc = tag.indexOf('>');
    int kraj = tag.lastIndexOf('<');
    String sadrzaj = tag.substring(poc + 1, kraj);
    System.out.println("Sadrzaj: " + sadrzaj);

    // ekstrakcija reci odvojenih zarezima
    String recen = "tamo,neka,recenica";
    String[] reci = recen.split(",");
    System.out.println("Reci:");
    for (String rec : reci)
      System.out.println(rec);

    // filtriranje svega sem cifara
    str = "abc{ 1a2--3def .4, ";
    str = str.replaceAll("[^0-9]", "");
    System.out.println("Cifre: " + str);
  }
}
```

Izlaz iz programa:

```
Trim-ovan: '1 2.4 56'
Sadrzaj: hello, world
Reci:
tamo
neka
recenica
Cifre: 1234
```

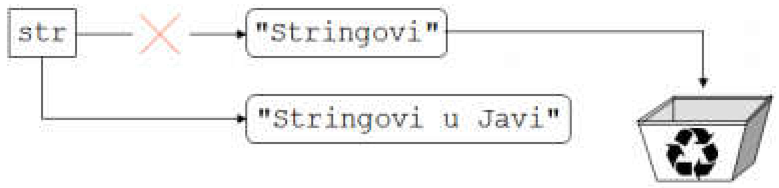
Nepromenljivost i performanse

- Objekti tipa String su nepromenljivi (immutable), što je osobina koja ima i prednosti i mane.
- Najveća prednost je što više referenci može bezbedno da pokazuje na isti objekat, jer nema opasnosti da promena sadržaja objekta preko jedne reference utiče na sadržaj koji se očekuje kad se koriste ostale reference.
- Glavna mana pristupa je u utrošku memorije i efikasnosti izvršavanja koda u kom se radi veliki broj izmena stringova, jer se za svaku izmenu kreira potpuno nova instanca tipa String.
- Ilustrujmo ovo na primeru dodavanja stringa na kraj postojećeg stringa:

```
String str = "Stringovi";
str += " u Javi";
```

Nepromenljivost i performanse

- Čak i kod ovako jednostavne operacije, inicijalni sadržaj na koji referencira promenljiva str („Stringovi“) se „ zaboravlja“, u smislu da se kreira potpuno novi objekat tipa String sa novim sadžajem „Stringovi u Javi“ i referenca na taj objekat se dodeljuje promenljivoj str.
- Na prethodni sadržaj („Stringovi“) prestaje da pokazuje ijedna referenca, što znači da je taj sadržaj spreman za dealokaciju i oslobađanje memorije od strane skupljača đubreta (garbage collector).



Slika 6.44: Nepromenljivost stringova za kod: `String str = "Stringovi";`
`str += " u Javi";`

Nepromenljivost i performanse

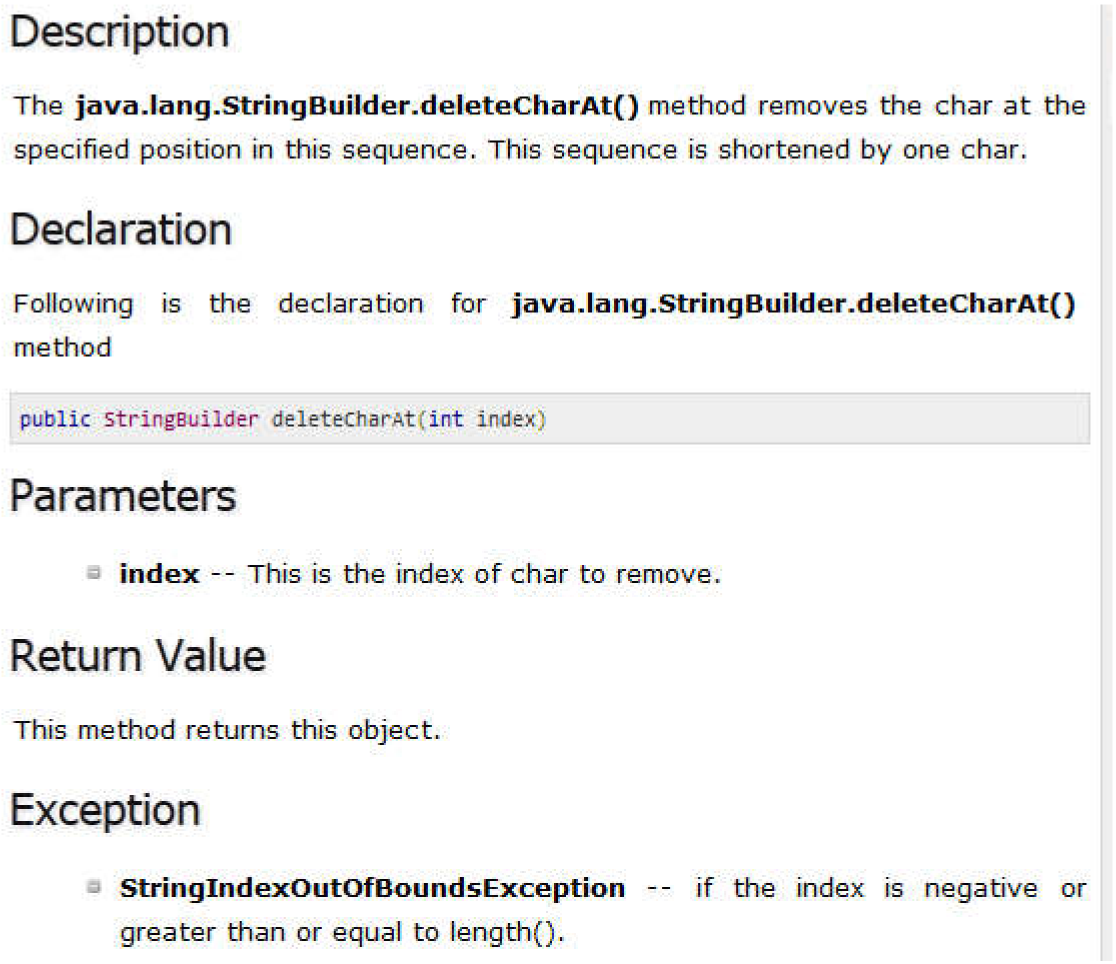
- Da bi se izbegla upotreba klase **`String`** u situacijama sličnim opisanoj , u Javi je uvedena klasa **`StringBuilder`**, koja predstavlja promenljivi (*eng.mutable*) ekvivalent klasi String.
- Glavne operacije koje klasa StringBuilder pruža a nisu dostupne u klasi String, implementirane su metodima **`append`** i **`insert`**, koje omogćavaju dodavanje sadržaja na kraj stringa i ubacivanje sadržaja unutar stringa, respektivno.
- Klasa StringBuilder takođe nudi metode slične kao u klasi String:
  - length,
  - indexOf,
  - lastIndexOf,
  - substring, itd.

Nepromenljivost i performanse

- Pored ovih, postoje novi metodi koji iskorišćavaju promenljivost sadržaja, na primer:
  - `StringBuilder deleteCharAt(int index)`: briše karakter na datoj poziciji u tekućem (this)StringBuilderu,
  - `StringBuilder delete(int start, int end)`: briše karaktere stringa definisanog datim pozicijama,
  - `StringBuilder reverse()`: obrće redosled karaktera,
  - `Void setCharAt(int index, Char, ch):` postavlja karaker na datoj poziciji na datu vrednost

Nepromenljivost i performanse

- `StringBuilder deleteCharAt(int index)`: briše karakter na datoj poziciji u tekućem (this)StringBuilderu.



Description

The **java.lang.StringBuilder.deleteCharAt()** method removes the char at the specified position in this sequence. This sequence is shortened by one char.

Declaration

Following is the declaration for **java.lang.StringBuilder.deleteCharAt()** method

```
public StringBuilder deleteCharAt(int index)
```

Parameters

- **index** -- This is the index of char to remove.

Return Value

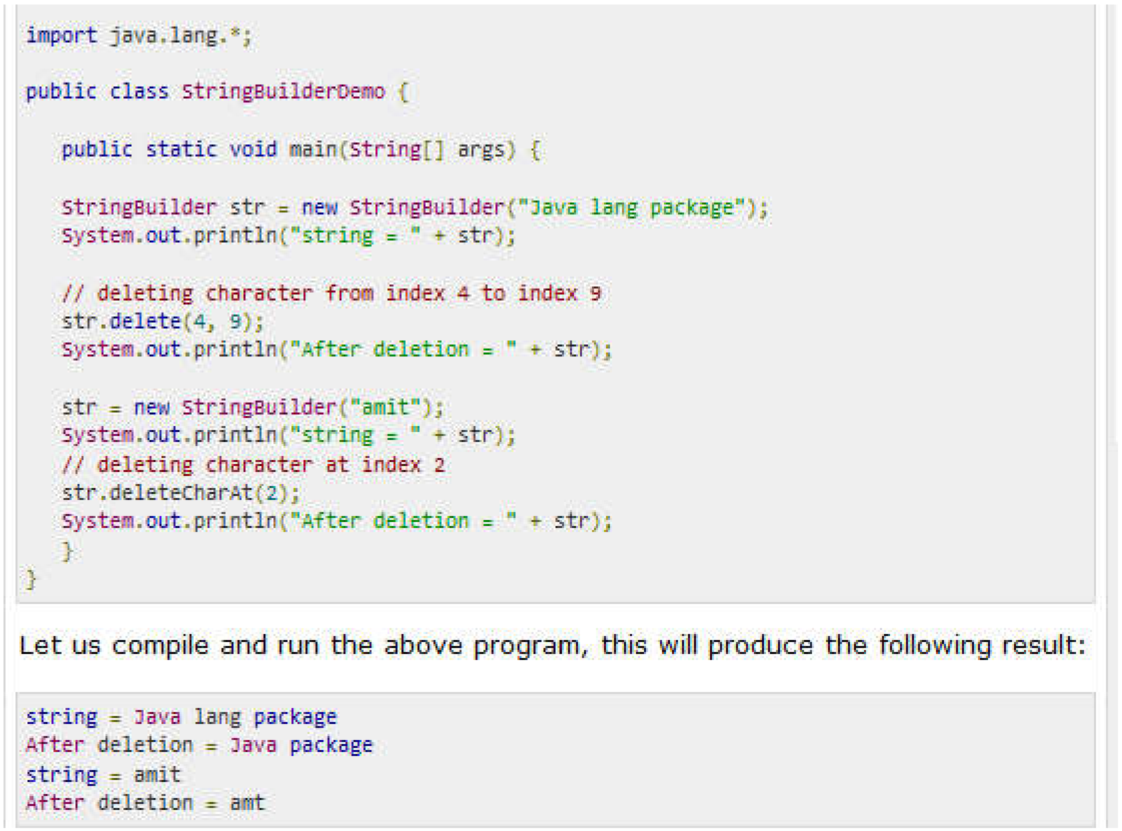
This method returns this object.

Exception

- **StringIndexOutOfBoundsException** -- if the index is negative or greater than or equal to length().

Nepromenljivost i performanse

- **`StringBuilder deleteCharAt(int index)`**: briše karakter na datoj poziciji u tekućem (this)StringBuilderu.



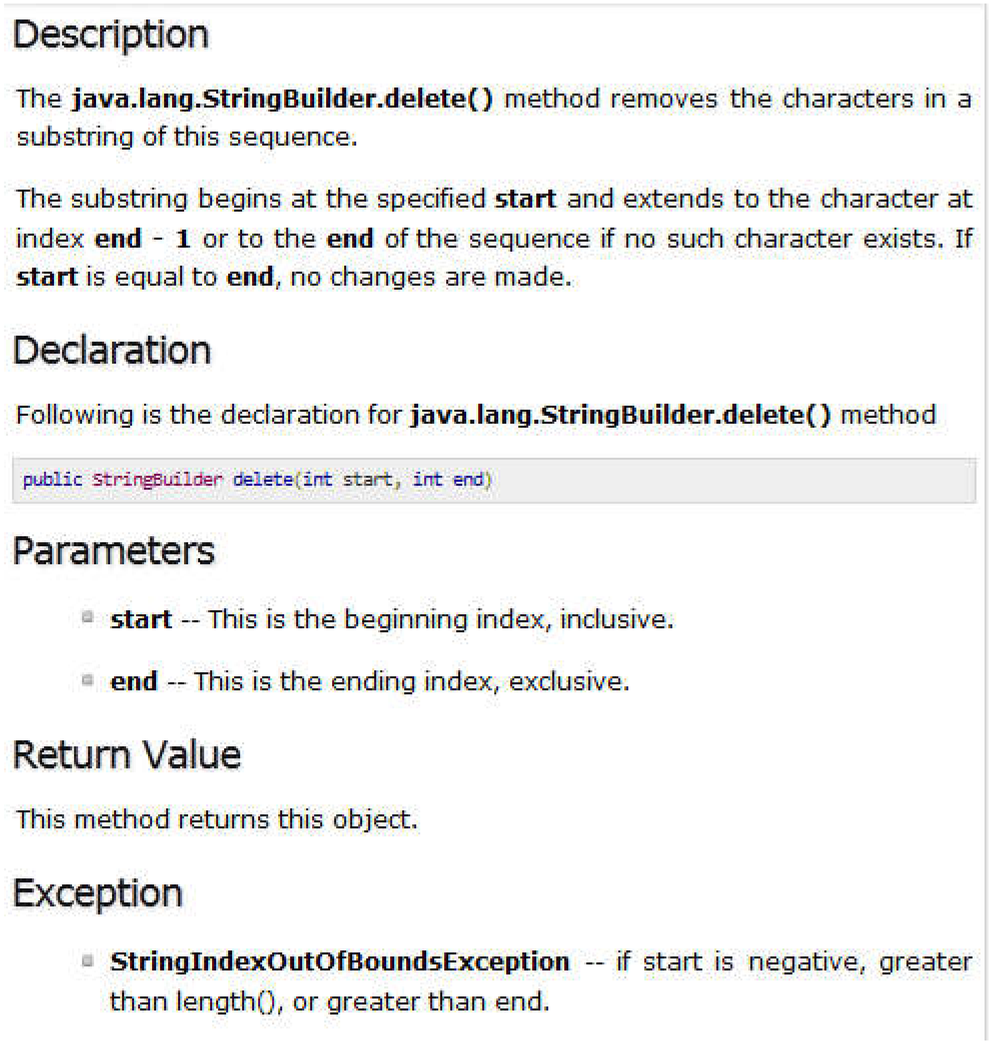
```
import java.lang.*;

public class StringBuilderDemo {

   public static void main(String[] args) {

   StringBuilder str = new StringBuilder("Java lang package");
   System.out.println("string = " + str);

   // deleting character from index 4 to index 9
   str.delete(4, 9);
   System.out.println("After deletion = " + str);

   str = new StringBuilder("amit");
   System.out.println("string = " + str);
   // deleting character at index 2
   str.deleteCharAt(2);
   System.out.println("After deletion = " + str);
   }
}
```

Let us compile and run the above program, this will produce the following result:

```
string = Java lang package
After deletion = Java package
string = amit
After deletion = amt
```

Nepromenljivost i performanse

– `StringBuilder delete(int start, int end)`: briše karaktere stringa definisanog datim pozicijama.



Description

The **java.lang.StringBuilder.delete()** method removes the characters in a substring of this sequence.

The substring begins at the specified **start** and extends to the character at index **end - 1** or to the **end** of the sequence if no such character exists. If **start** is equal to **end**, no changes are made.

Declaration

Following is the declaration for **java.lang.StringBuilder.delete()** method

```
public StringBuilder delete(int start, int end)
```

Parameters

- **start** -- This is the beginning index, inclusive.
- **end** -- This is the ending index, exclusive.

Return Value

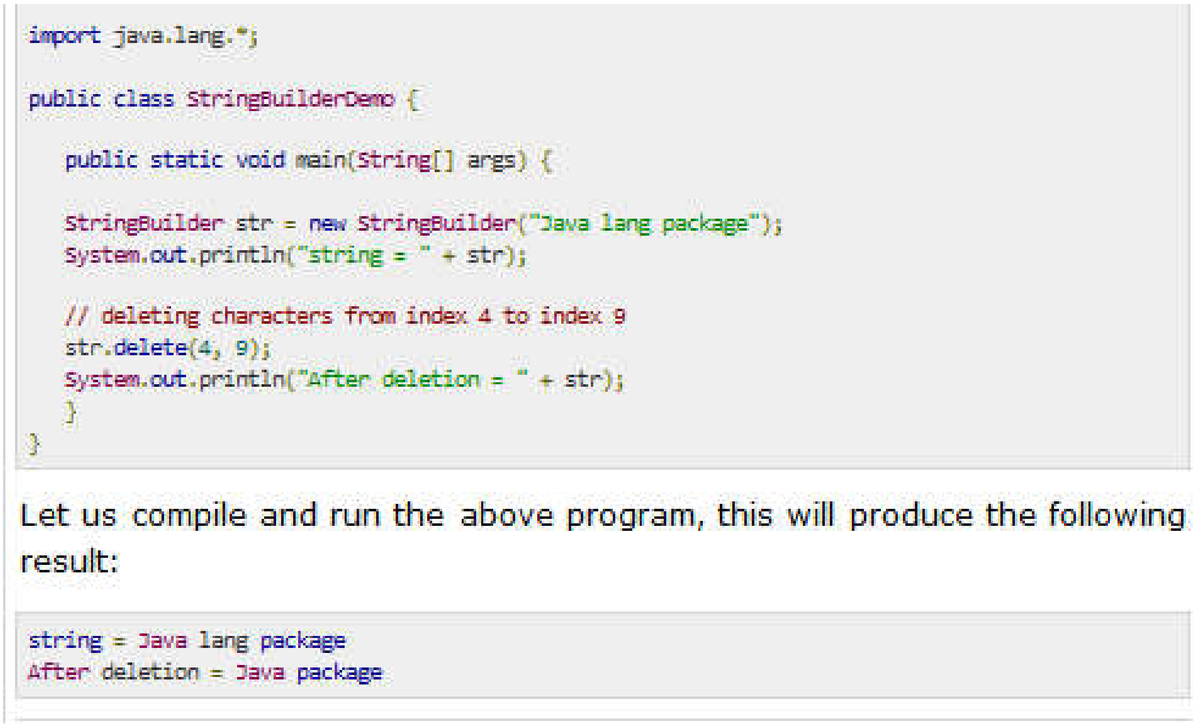
This method returns this object.

Exception

- **StringIndexOutOfBoundsException** -- if start is negative, greater than length(), or greater than end.

Nepromenljivost i performanse

– **`StringBuilder delete(int start, int end)`:** briše karaktere stringa definisanog datim pozicijama.



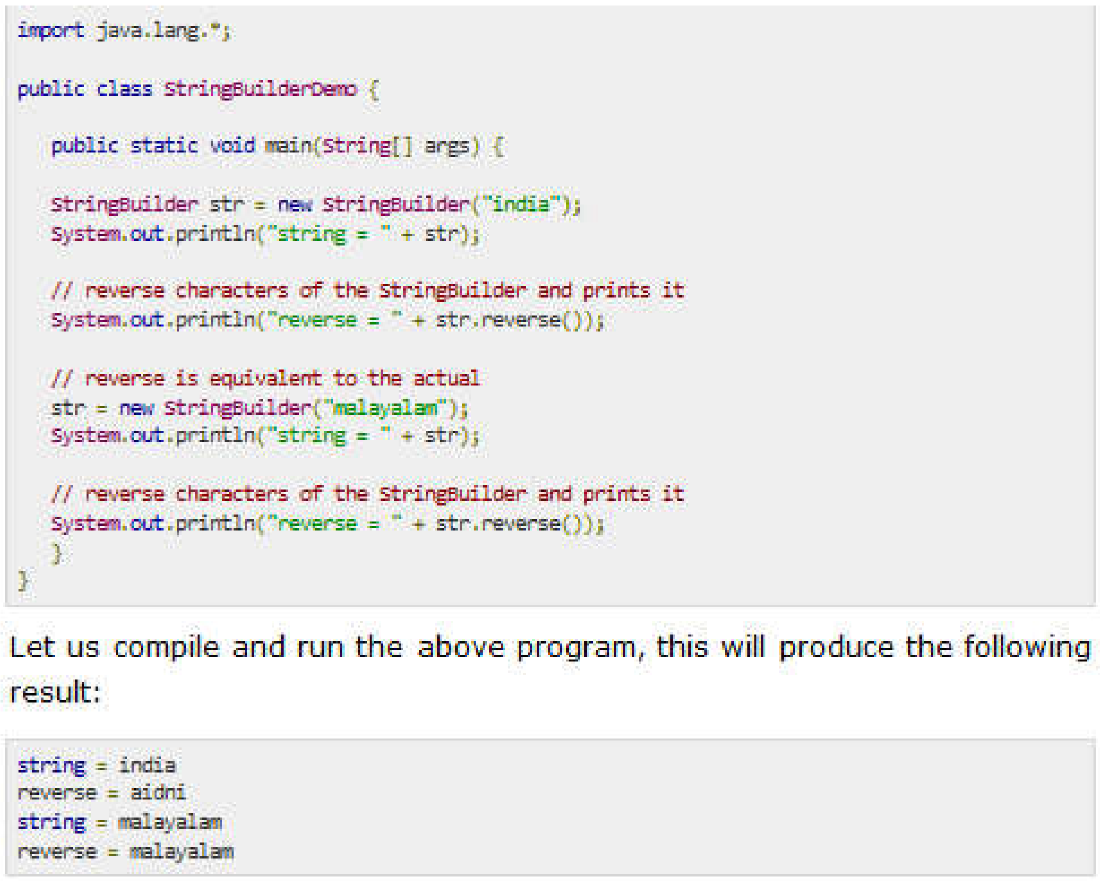
```
import java.lang.*;

public class StringBuilderDemo {

   public static void main(String[] args) {

   StringBuilder str = new StringBuilder("Java lang package");
   System.out.println("string = " + str);

   // deleting characters from index 4 to index 9
   str.delete(4, 9);
   System.out.println("After deletion = " + str);
   }
}
```

Let us compile and run the above program, this will produce the following result:

```
string = Java lang package
After deletion = Java package
```

Nepromenljivost i performanse

– `StringBuilder reverse()`: obrće redosled karaktera,



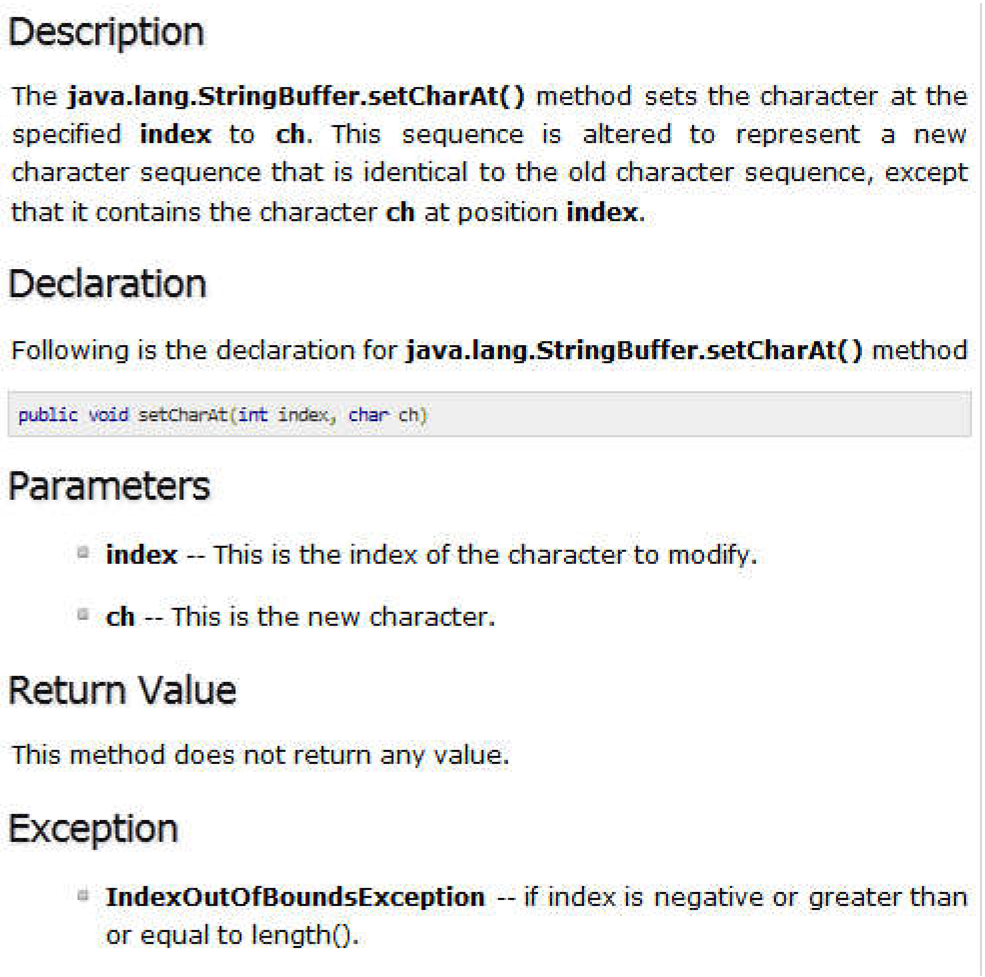
```
import java.lang.*;

public class StringBuilderDemo {

   public static void main(String[] args) {

   StringBuilder str = new StringBuilder("india");
   System.out.println("string = " + str);

   // reverse characters of the StringBuilder and prints it
   System.out.println("reverse = " + str.reverse());

   // reverse is equivalent to the actual
   str = new StringBuilder("malayalam");
   System.out.println("string = " + str);

   // reverse characters of the StringBuilder and prints it
   System.out.println("reverse = " + str.reverse());
   }
}
```

Let us compile and run the above program, this will produce the following result:

```
string = india
reverse = aidni
string = malayalam
reverse = malayalam
```

Nepromenljivost i performanse

– **Void setCharAt(int index, Char, ch):** postavlja karaker na datoj poziciji na datu vrednost



Description

The **java.lang.StringBuffer.setCharAt()** method sets the character at the specified **index** to **ch**. This sequence is altered to represent a new character sequence that is identical to the old character sequence, except that it contains the character **ch** at position **index**.

Declaration

Following is the declaration for **java.lang.StringBuffer.setCharAt()** method

```
public void setCharAt(int index, char ch)
```

Parameters

- **index** -- This is the index of the character to modify.
- **ch** -- This is the new character.

Return Value

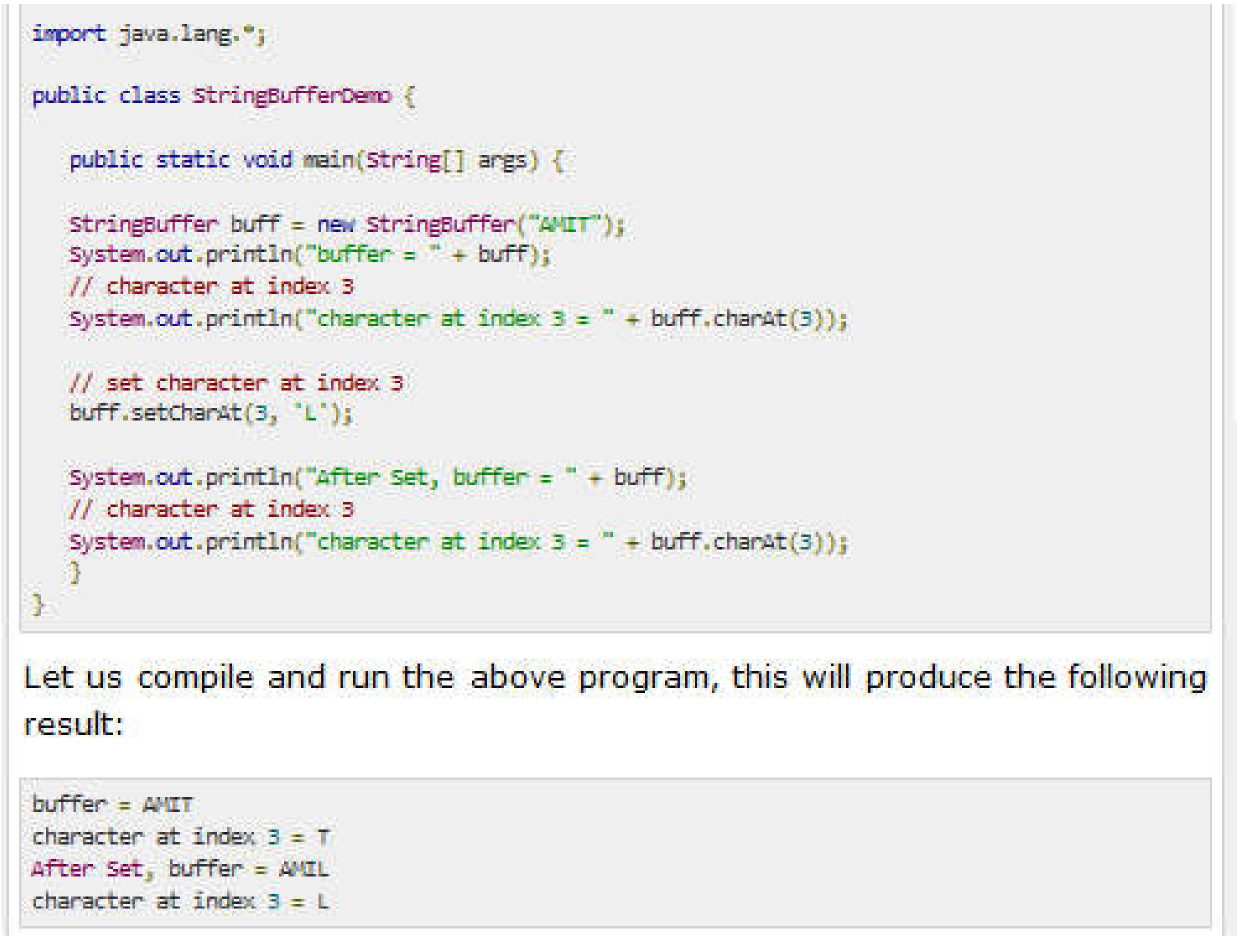
This method does not return any value.

Exception

- **IndexOutOfBoundsException** -- if index is negative or greater than or equal to length().

Nepromenljivost i performanse

– **`Void setCharAt(int index, Char, ch):`** postavlja karaker na datoj poziciji na datu vrednost



```
import java.lang.*;

public class StringBufferDemo {

   public static void main(String[] args) {

   StringBuffer buff = new StringBuffer("AMIT");
   System.out.println("buffer = " + buff);
   // character at index 3
   System.out.println("character at index 3 = " + buff.charAt(3));

   // set character at index 3
   buff.setCharAt(3, 'L');

   System.out.println("After Set, buffer = " + buff);
   // character at index 3
   System.out.println("character at index 3 = " + buff.charAt(3));
   }
}
```

Let us compile and run the above program, this will produce the following result:

```
buffer = AMIT
character at index 3 = T
After Set, buffer = AMIL
character at index 3 = L
```

Nepromenljivost i performanse

- Za razliku od metoda klase `String` koje vraćaju referencu na novu instancu tipa String, metode klase `StringBuilder` sa povratnim tipom `StringBuilder` vraćaju nepromenjenu referencu na tekući objekat čiji je sadržaj promenjen.

- Ako se u programu očekuju česte promene sadržaja stringova, zbog opisanih osobina mnogo je bolje koristiti klasu `StringBuilder` nego `String` kako zbog brzine izvršavanja tako i zbog bolje iskorišćenosti memorije.

Nepromenljivost i performanse

- Takođe, moguće je podesiti kapacitet **`StringBuilder-a`** u odgovarajućem konstruktoru, čime se unapred može zauzeti memorija potrebna za izvršenja nekog zadatka.

- Ako se kapacitet **`StringBuilder`**.a u toku izvršavanja programa premaši, on će se automatski proširiti, što je spora operacija.

- Tako da je prilikom pravljenja novih instanci klase **`StringBuilder`** i postavljanja inicijalnih kapaciteta potrebno pronaći ravnotežu između obezbeđenja dovoljno memorije da se problem reši i zauzimanja previše memorije što program može učiniti prezahtevnim.

Nepromenljivost i performanse

- Sledeći primer ilustruje razliku u brzini izvršavanja jednostavne operacije spajanja velikog broja kopija jednog stringa korišćenjem klasa `String` i `StringBuilder`.

- U metodu `concatString`, osnovni string **OSNOVA** dodaje se na kraj String-a `rez` pomoću operatora `+=`, pri čemu se svaki put konstruiše potpuno nova instanca stringa.

- S druge strane, u metodu `concatBuilder` se koristi `StringBuilder` inicijalizovan sa odgovarajućim kapacitetom, u koji se redom dodaju kopije stringa OSNOVA pomoću metoda `append`.

Primer 6.83: Test brzine spajanja stringova pomoću klasa `String` i `StringBuilder`

```
public class StringSpeedTest {
  private static final String OSNOVA = "Tri slepa misa... jure kao ludi
      ...";

  // spajanje stringa OSNOVA 'broj' puta
  private static String concatString(int broj) {
    String rez = "";
    for (int i = 0; i < broj; i++)
      rez += OSNOVA;
    return rez;
  }

  // dodavanje stringa OSNOVA 'broj' puta
  // (koriscenjem StringBuilder-a)
  private static String concatBuilder(int broj) {
    int kapacitet = OSNOVA.length() * broj;
    StringBuilder builder = new StringBuilder(kapacitet);
    for (int i = 0; i < broj; i++)
      builder.append(OSNOVA);
    return builder.toString();
  }

  public static void main(String[] args) {
    final int BROJ = 100000;

    long poc = System.nanoTime();
    concatString(BROJ);
    long trajanje = System.nanoTime() - poc;
    double vremeStr = trajanje / 1000000000.0;

    poc = System.nanoTime();
    concatBuilder(BROJ);
    trajanje = System.nanoTime() - poc;
    double vremeSB = trajanje / 1000000000.0;

    System.out.println("Vreme za String: " + vremeStr + "s");
    System.out.println("Vreme za StringBuilder: " + vremeSB + "s");
  }
}
```

Izlaz iz programa:

```
Vreme za String: 120.628253047s
Vreme za StringBuilder: 0.009280778s
```